REFRESH REDUCTION IN DYNAMIC MEMORIES

BY

ADITYA AGRAWAL

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Doctoral Committee:

Professor Josep Torrellas, Chair
Professor Sanjay Patel
Professor Naresh Shanbhag
Dr. Dinesh Somasekhar, Intel Corp.

ABSTRACT

An effective approach to reduce the static energy consumption of large on-chip memories is to use a low-leakage technology such as embedded DRAM (eDRAM). Unfortunately, eDRAM, being a dynamic memory, requires periodic refresh, which ends up consuming substantial energy for large last-level caches.

In upcoming architectures that stack a processor die and multiple DRAM dies, DRAM dies experience higher temperatures. Elevated temperatures increase the periodic refresh requirement of DRAM, also a dynamic memory, which increases its energy and hurts processor performance. In this thesis, we propose and evaluate techniques for refresh reduction in dynamic memories.

We examine the opportunity of refreshing only the data that will be used in the near future and only if the data has not been recently accessed (and automatically refreshed). We present *Refrint*, a simple approach to perform fine-grained refresh of on-chip eDRAM multiprocessor cache hierarchies. We show that an eDRAM-based memory hierarchy with Refrint consumes only 30% of the energy of a conventional SRAM-based memory hierarchy, and induces a slowdown of only 6%. In contrast, an eDRAM-based memory hierarchy without Refrint consumes 56% of the energy of the conventional memory hierarchy, inducing a slowdown of 25%.

While it is well known that different eDRAM cells exhibit very different charge-retention properties, current systems pessimistically assume worst-case retention times, and refresh all the cells at a conservatively-high rate. We use known facts about the factors that determine the retention properties of cells to build a new model of eDRAM retention times. The model is called *Mosaic*. We show that the retention times of cells in large eDRAM modules exhibit substantial spatial correlation. We propose a mechanism to exploit such correlation to save refresh energy. With simple techniques, we reduce the refresh energy of large eDRAM modules by 20x. The result is that refresh energy is all but eliminated.

Finally, we focus on temperature and refresh reduction in 3D processor-memory

stacks. We propose the *Xylem* Thermal Through Silicon Via (TTSV) placement schemes, to reduce the temperature of the DRAM dies and the processor die in the stack. The TTSV placement should respect the DRAM array structure and handle unknown multicore hotspots. The resulting temperature distribution still has a significant spatial variation. Therefore, we propose new DRAM refresh schemes that take advantage of this spatial variation in temperature. Our best Xylem TTSV placement scheme reduces the peak temperature of the DRAM stack by an average of 8.7 $^{\circ}$C. Combined with the best DRAM refresh scheme we reduce the number of refreshes by an average of 85%.

To my parents, sisters Radhika and Rashmi and my wife Smriti
for their boundless love, support and encouragement.

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to my advisor, Prof. Josep Torrellas, for his guidance, support and encouragement throughout my doctoral work. He played a pivotal role during idea generation and developing insights. In spite of his busy schedule he was always available for discussion. He is very hard working, organized, a perfectionist, knowledgeable yet soft spoken and is a role model for me. His zeal for research and technological advancement is contagious. He gave me a lot of academic freedom but at the same time always nudged and steered me in the correct direction. If I could turn back time to the start of my doctoral journey, I would still pick him to be my advisor.

I would also like to thank my coauthors, Amin Ansari and Prabhat Jain, and my labmates, Bhargava Reddy and Raghavendra Pothukuchi, for the numerous brainstorming sessions which helped me bring my ideas to fruition.

I am fortunate enough to have worked closely with David Dunning, my manager and mentor during the many internships at Intel. He has been a great influence. His pursuits in car racing, kayaking, wood carving in addition to his technical expertise is an example of work life balance I can only hope to achieve.

No journey is easy without good friends, and mine has been in the company of Rakesh Komuravelli, Ravi Tumkur and Sushil Kumar. We shared the crests and troughs of doctoral student life and had a great time.

Last but not the least, all I have been able to do and achieve is because of the love and support of my parents, my sisters Radhika and Rashmi, and my wife Smriti. My parents and sisters were very supportive of my dream to pursue doctoral studies from the beginning. Along the way, I met Smriti who renewed my energy and enthusiasm to achieve my dream.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

While CMOS technology continues to enable a higher integration of transistors on a chip, energy, power and temperature have emerged as the true constraints for more capable systems. Unfortunately, a significant fraction of the energy consumed by a large processor chip is actually wasted away as leakage. In particular, since memory modules often use a big fraction of the chip's real estate, leakage in the on-chip memory hierarchy is a major source of energy waste.

An approach to reduce on-chip memory leakage is to use a memory technology that, while compatible with a logic process, does not leak by design — or leaks much less. One obvious example is embedded DRAM (eDRAM). The IBM POWER7 processor, for example, uses eDRAM for the on-chip L3 cache [1].

Roughly speaking, for the same size in Kbytes as SRAM, eDRAM reduces the leakage power to an eighth or less [2]. It has the shortcoming that it needs to be refreshed, and refresh power is significant [3]. However, this fact in turn offers a new opportunity for power savings, through fine-grained refresh management. Another shortcoming of eDRAM is its lower speed but, arguably, processors will be able to tolerate this issue for non-L1 caches.

Maturing technology for Through Silicon Vias (TSVs) [4, 5, 6, 7] is about to enable further integration of computer architectures into 3D processor-memory stacks. In these architectures, on-chip temperatures will be higher than in conventional planar designs because of the increased transistor density and the higher inter-layer thermal resistance. It is well known that high temperatures are taxing for DRAM. The data retention decreases and, therefore, cells need more frequent refresh. Therefore, more aggressive refresh schemes are needed, to avoid high energy consumption and throughput loss.

Intuitively, the large on-chip multi-level cache hierarchy of a manycore is likely to contain much useless (or dead) data. Keeping such data on chip results in unnecessary refresh. In our first proposal, *Refrint*, we examine the opportunity of power savings by intelligently refreshing an on-chip eDRAM cache hierarchy.

Our goal is to refresh only the data that will be used in the near future and only if the data has not been recently accessed. The other data is invalidated and/or written back to main memory.

Recent experimental work from IBM has shown that the retention time of an eDRAM cell strongly depends on the threshold voltage (V_t) of its access transistor [8]. In our second proposal, *Mosaic*, we note that, since the values of V_t within a die have a strong spatial correlation, then eDRAM retention times will also necessarily exhibit spatial correlation. Consequently, in this proposal, we first develop a new model of the retention times in large on-chip eDRAM modules. Then, based on the model, we develop simple architectural mechanisms to exploit such correlation to eliminate much of the refreshes at low cost.

In *Xylem*, we address the problem of thermal and refresh management in 3D processor-memory stacks. Thermal TSVs are dummy TSVs and are meant for thermal conduction as opposed to electrical conduction. They take heat away from the active parts of the die. We propose thermal TSV placement schemes that are effective in reducing the temperature of the dies in the stack. In addition, given the temperature distribution, we find that existing refresh schemes for planar DRAM technologies are inadequate in this environment. Therefore, we also propose new DRAM refresh schemes for these architectures to reduce the number of refreshes.

This document is organized as follows: Chapter 2 covers the architecture, implementation, evaluation and related work for Refrint; Chapter 3 describes the Mosaic retention time model, its architecture, evaluation and related work, and Chapter 4 covers the Xylem thermal TSV placement schemes, refresh schemes, evaluation and related work.

CHAPTER 2

REFRINT

In this chapter, we examine the opportunity for power savings by intelligently refreshing an on-chip eDRAM cache hierarchy. Our goal is to refresh only the data that will be used in the near future, and only refresh it if it is really needed. The other data is invalidated and/or written back to main memory. We present *Refrint*, a simple approach for fine-grained, intelligent refresh of eDRAM lines to minimize on-chip power. We introduce the Refrint algorithms and the microarchitecture support required.

Our results show that Refrint is very effective. We evaluate 16-threaded parallel applications running on a simulated manycore with a three-level cache hierarchy, where L2 and L3 can be SRAM or eDRAM. The eDRAM-based memory hierarchy with Refrint consumes only 30% of the energy of a conventional SRAM-based memory hierarchy. In addition, Refrint's early invalidation and writeback of lines only increases the execution time of the applications by 6%. In contrast, an eDRAM-based memory hierarchy without Refrint consumes 56% of the energy of the conventional memory hierarchy, and induces an application execution slowdown of 25%.

This chapter is organized as follows: Section 2.1 provides the motivation; Sections 2.2 and 2.3 present the architecture and implementation of Refrint; Sections 2.4 and 2.5 evaluate Refrint; and Section 2.6 covers related work.

2.1 Motivation

Current trends suggest a progressive increase in the number of cores per chip in the server market. These cores need to be relatively simple to meet the power and thermal budget requirements of chips. Moreover, a large fraction of the area is regularly devoted to caches — in fact, more than 70% in Niagara [9]. In addition, these chips are including extensive clock gating and ever more sophisticated

management techniques for dynamic power. There is even significant interest in reducing the supply voltage of the chip and clocking it at moderate frequencies, to operate in a much more energy-efficient environment [10], with lower dynamic power.

The combination of progressively lower dynamic power and large on-chip caches points to on-chip cache leakage as one of the major contributors to present and, especially, future chip power consumption [11]. As a result, there have been proposals for new approaches and technologies to deal with on-chip SRAM leakage. These proposals include power gating (e.g., [12, 13, 14, 15, 16]), new SRAM organizations [11, 17], embedded DRAM (eDRAM), on-chip flash, and non-volatile memory technologies. Section 2.6 discusses this work.

One of the most interesting proposals is eDRAM, which has been used by IBM in the 32MB last level cache (LLC) of the POWER7 processor [1]. eDRAM is a capacitor-based dynamic RAM that can be integrated on the same die as the processor. Compared to the SRAM cell, eDRAM has much lower leakage and a higher density. It also has a lower speed. However, as we explore lower supply voltages and frequencies for energy efficiency, eDRAM may be a very competitive technology for non-L1 caches.

A major challenge in using eDRAM as the building cell of on-chip caches is its refresh need. Since eDRAM is a dynamic cell, it needs to be refreshed at periodic intervals called *retention periods*, to preserve its value and prevent decay. On an access, the cell automatically gets refreshed, and stays valid for another retention period. Overall, refreshing the cells imposes a significant energy cost, and may hurt performance because the cells are unavailable as they are being refreshed.

It is well known that today's large last-level caches of chip multiprocessors contain a lot of useless data. There is, therefore, an opportunity to minimize the refresh energy by not refreshing the data that is not useful for the program execution anymore. The challenge is to identify such data inexpensively.

2.2 Refrint Architecture

2.2.1 Main Idea

We consider an on-chip multiprocessor cache hierarchy where L2 and L3 use eDRAM. We identify two sources of unnecessary refreshes, as summarized in

$$\underset{Refreshes}{Unnecessary} = \underset{(lower\ level)}{Cold\ lines} + \underset{(upper\ level)}{Hot\ lines}$$

Figure 2.1: Sources of unnecessary refreshes.

Fig. 2.1. The first are *Cold* lines, namely those that are not being used or are being used far apart in time, but are still getting refreshed (Fig. 2.2). The second are *Hot* lines, which are those that are actively being used but are still getting refreshed because of the naive periodic refresh policy in eDRAMs (Fig. 2.3). Recall that, on a read or a write, a line is automatically refreshed, and hence there is no need to refresh it for another retention period. Cold lines are typically found in lower-level caches such as L3, while hot lines may be found in upper-level caches closest to the processor, such as L2.

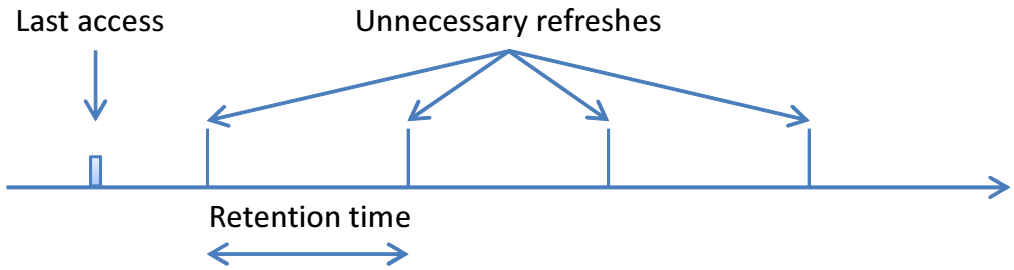


Figure 2.2: Access pattern of a cold line.

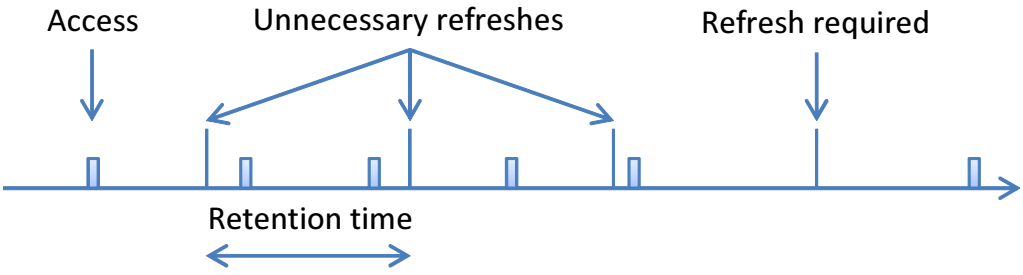


Figure 2.3: Access pattern of a hot line.

For the cold lines, we propose "data-based" policies, which identify and refresh only those cache lines which are expected to be used again in the near future. The rest of the lines are invalidated from the cache and not refreshed.

If the policies are too aggressive, we may end up invalidating a lot of useful lines from the caches, thereby having to access the lower level memory hierarchy

to refetch lines a far higher number of times than in a conventional cache hierarchy. Also, writing back and invalidating a soon-to-be-accessed dirty line has double the penalty of invalidating a soon-to-be-accessed clean line, as it involves writing back the dirty line. Therefore, our policies need to be more conservative at handling dirty lines.

For the Hot lines, we propose "time-based" policies, which try to avoid refreshing lines after they have been accessed (and automatically refreshed). They improve over a naive periodic scheme that eagerly refreshes a line at regular periods, oblivious of when the line was last accessed.

In this proposal, we focus on simple refresh policies. We do not consider line reuse predictors or similarly elaborate hardware structures. Also, we do not assume that we have information provided by the programmer or software system.

2.2.2 Refresh Policies Proposed

A refresh policy has a time- and a data-based component (Table 2.1). The time-based component decides when to refresh, while the data-based one decides what to refresh.

Table 2.1: Refresh policies proposed.

Time-based policies: *When ?*	
Periodic	Refresh periodically
Polyphase	Refresh in the same phase
Data-based policies: *What ?*	
All	Refresh all lines
Valid	Refresh only Valid lines
Dirty	Refresh only Dirty lines
WB(n,m)	Refresh idle Dirty lines *n* times before writeback and refresh idle Valid Clean lines *m* times before invalidation

2.2.2.1 Time-Based Policy

We propose a time-based policy called *Polyphase*. Polyphase divides the retention period into a fixed number of equal intervals called *Phases*. Each cache line maintains information on which phase the line was last accessed. The phase information is updated on every read or write to the line. With Polyphase, a line

is refreshed only when the same phase arrives in the next retention period. This approach reduces the number of refreshes of hot cache lines.

Fig. 2.4 shows an example of Polyphase, where a retention time is divided into four phases. A line is accessed in Phase 2 of the first retention time. Hence, rather than refreshing it again when the current retention period expires, it is refreshed only at the beginning of Phase 2 of the next retention period.

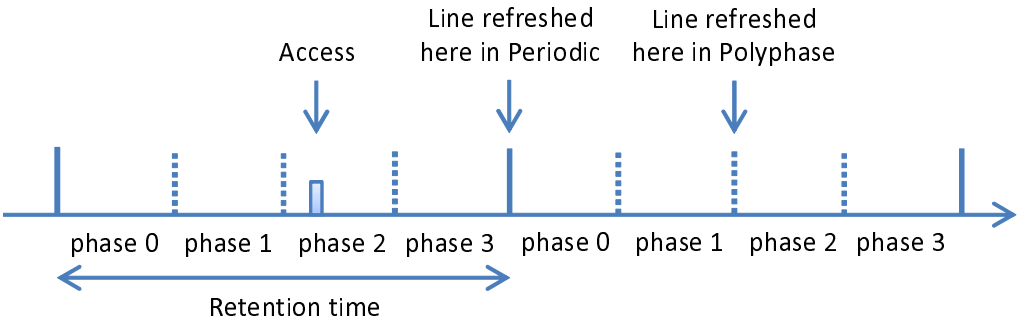


Figure 2.4: Polyphase with 4 phases.

In Polyphase, at the beginning of each phase, the cache controller quickly checks for lines whose phase information matches that of the controller. The phase information of each cache line is maintained in the cache controller and not in the data array. Hence, the check does not block the data array. The lines with the current phase are either refreshed or invalidated, depending on the data-policy employed. Every access to a cache line refreshes the cache line and updates its phase information. This approach performs the minimum number of refreshes to keep a particular line alive.

We also consider the trivial *Periodic* policy. In this case, the cache controller refreshes lines at periodic intervals equal to the retention period of the eDRAM cells. This approach is cheap because it requires no phase bits; it only needs a global counter for the whole cache. However, it results in more refreshes than necessary, since it may eagerly refresh a line long before it is about to decay. Also, it may render the cache unavailable for a continuous period of time, when the lines are being refreshed.

2.2.2.2 Data-Based Policy

We propose the simple data-based policies shown in Table 2.1. They consider the state of the cache line in a multiprocessor hierarchy (valid, dirty, etc.), to decide what to refresh.

Specifically, our policies are *All*, *Valid*, *Dirty*, and *WB(n,m)* (for write back). *All* refreshes every cache line, irrespective of whether it is valid or not. We evaluate this policy only for reference purposes. *Valid* and *Dirty* refresh Valid and Dirty cache lines, respectively, and invalidate the line otherwise.

The *WB* policy is associated with a tuple *(n,m)*. WB refreshes a Dirty line that is not being accessed for *n* times before writing it back and changing its state to Valid Clean; moreover, it refreshes a Valid Clean line that is not being accessed for *m* times before invalidating it. WB retains a Dirty line in the cache longer because evicting it has the additional cost of writing the line back to lower-level memory. To implement WB, we maintain a per-line *Count*. When the line is read or written, Count is set to n (if Dirty) or m (if Valid Clean). In addition, when the line is refreshed, Count is decremented. When Count reaches zero, the line is either written back or invalidated. Note that the *Dirty* policy is equivalent to *WB(∞,0)*, while *Valid* is equivalent to *WB(∞,∞)*. Finally, every policy refreshes cache lines in transient states as well.

Using cache line states for the refresh policy has the advantage that the hardware needed is simple. A disadvantage is that the policy is unable to disambiguate same-state lines that behave differently. In addition, the policy interacts with the cache coherence protocol and the inclusivity properties of multilevel caches. For example, if a policy decides to invalidate a line from L3, due to cache inclusivity, it also has to invalidate the line from L2 and L1. This results in extra messages.

In our analysis and evaluation, we use the Periodic time-based policy and the All data-based policy as the baseline policy. A slightly smarter and natural extension is the Periodic Valid policy.

2.2.3 Application Categorization

We now categorize applications based on how they are affected by our time and data-based policies. We assume a cache-coherent multiprocessor with a multilevel, inclusive on-chip cache hierarchy, where the last level is shared by all the cores.

2.2.3.1 Time-Based Policy

Polyphase is effective in reducing the number of refreshes to a line when the interval between accesses to the line is shorter than its retention time — i.e., the frequency of accesses is higher than the refresh rate (1/Retention Time). This is shown in Fig. 2.5.

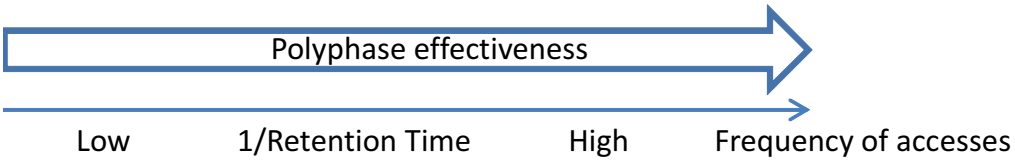


Figure 2.5: Application categorization according to the time-based policy.

For cache levels close to the processor, the frequency of cache accesses is high, but the fraction of energy consumed by refreshing is small. Hence, the overall impact of the time-based policy tends to be small. For cache levels far from the processor such as L3, the fraction of energy consumed by refreshing is high. However, we do not typically see repeated accesses to the same line because most of the accesses are intercepted by upper-level caches.

There are a few cases when lines in lower-level caches can observe repeated accesses. One is in applications with fine-grained sharing, where repeated coherence activity induces frequent writebacks to and reads from the shared cache. A second one is in codes with significant conflicts in the upper-level caches. A third one is in codes with accesses that are required to bypass the upper-level caches. Overall, in the application set that we evaluate in this proposal, we do not find codes where any of these behaviors is dominant.

2.2.3.2 Data-Based Policy

We are interested in observing the effects from the point of view of the last level cache, which is the one that matters the most in the total refresh energy consumed. Fig. 2.6 presents an application categorization based on two axes: application footprint and visibility.

The X axis shows the size of the application footprint relative to the size of the last level cache. Since an application can only access the data at a given maximum rate, it is likely that applications that have a large data footprint will have long time intervals between reuse of the data, if data is reused at all. Hence, lines can be

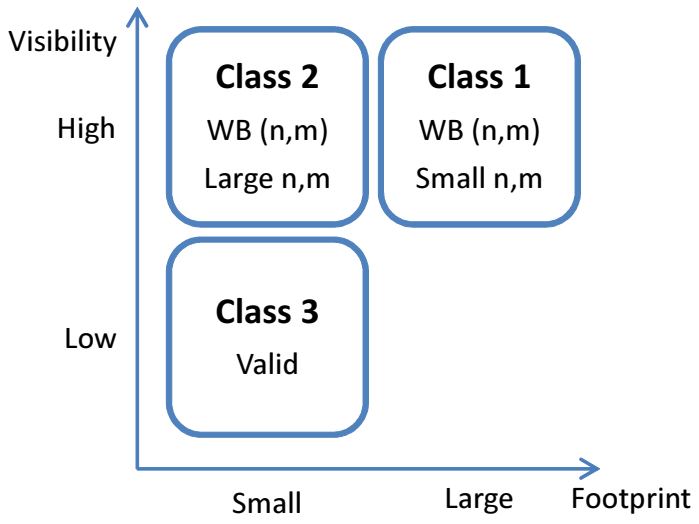


Figure 2.6: Application categorization according to the data-based policy.

written back and/or invalidated; if they are reused, they can be brought back again. Therefore, the best policy for such applications should be a general one, such as WB(n,m), with a *small* (n,m). After an initial flurry of accesses, the data can soon be evicted from the cache. On the other hand, in small-footprint applications, the processors are likely to reuse the data more often. Therefore, a general policy such as WB(n,m), with a *large* (n,m), is likely to be useful. Data will be reused and, therefore, should be kept in the on-chip memory.

The Y axis captures the last level cache's "visibility" on the activity of the lines in the upper levels of the cache hierarchy. For example, assume that the working set is such that: (i) it largely fits in the L1 and L2 caches (hence, there is no overflow) and (ii) there is little data sharing between processors (hence, the shared L3 cache does not see the data moving back and forth between caches and its associated state transitions between Dirty and Valid Clean). In this case, L3's visibility is low. Therefore, we need to be conservative and assume that the data is being repeatedly accessed in the L1 and L2 caches. Hence, the conservative Valid policy should be best for such applications, as it avoids invalidating data that could potentially be heavily reused in upper-level caches.

Even with a small data footprint, if there is high data sharing between cores, such that data is frequently written by a processor and read by another, then visibility at L3 is high. There are frequent writebacks to and reads from L3. In such cases, a more specific policy such as WB(n,m) should do better than Valid.

We refer to the three classes of applications described as Class 1, Class 2, and Class 3 applications, respectively. In our application set, we do not find any code of the type Large footprint and Low visibility; all the large-footprint applications also provide visibility to L3. This is either because they have substantial data sharing between cores, or because dirty data is often evicted from upper-level caches and written back to L3, therefore providing visibility.

2.3 Implementation Issues

In this section, we introduce the concepts of global and local phases, their implementation, and their use.

2.3.1 Phase Bits Design & Operation

We statically divide the retention time T into 2^N parts called *Global Phases*. For example, if $N = 2$, the retention time is divided into four global phases. The first quarter is the first phase, the second quarter is the second phase, and so on. We need N bits to encode 2^N global phases. In addition, each cache line is associated with N bits, called *Local Phase* bits. These bits record the global phase when the line was read or written. Hence, the number of global and local phase bits is the same. From our experiments, we found that values of 1 or 2 for N are good.

Assume that the system has an M bit counter ($M > N$) to support the retention time, such that 2^M clock ticks are equal to T. If we divide the retention time into 2^N phases, the N most significant bits of the counter will indicate the global phase. The counter is part of the baseline eDRAM implementation to keep track of retention time, and so there is no extra cost associated with keeping track of the global phase information.

Detecting a global phase transition is easy. Whenever the $M - N$ least significant bits of the counter become zero, we can infer a global phase transition.

For each cache line, we store the *N* local phase bits and a copy of the valid bit in the cache controller, in a structure called the *Phase Array*. If we assume a cache with a line size of 64 bytes and use *N*=2, the overhead in a line is 3 bits per 512 bits, which is less that 0.6%. If we assume a cache of size 1 MB, we have 16,384 lines. This requires a Phase Array of 16,384 $\times$ 3 bits = 49,152 bits = 6 KBytes.

Such phase array can be organized as 96 rows of 64 Bytes each. Each row (line) then holds information for $\approx$ 171 lines of the cache.

On a normal read or write access (and hence automatic refresh) to a cache line, the global phase bits are copied to the local phase bits. This allows us to keep track of the global phase in which the line was accessed. If the same line is not accessed again, we will not refresh it for another retention period, i.e., until the same global phase in the next retention period. However, if the line is accessed before then, the local phase information is updated, and a refresh does not happen for another retention period beginning from that phase.

At the beginning of each global phase, all normal read and write requests are put on hold. The cache controller quickly scans the phase bits of the valid cache lines and, if their local phase bits match the global phase bits, schedules the line for refresh. Recall that the phase information and a copy of the valid bit is maintained inside the cache controller. From the example above, reading just one line of the array provides information for 171 lines. Thus, the cache controller can quickly find the lines that need to be refreshed, and issue back to back refresh requests. Finally, the controller releases the hold on normal accesses.

The phase array design and the associated logic is shown in the upper half of Fig. 2.7. The actions at the beginning of each global phase are summarized below.

```
hold all read and write requests to the data array
for (all the lines of the cache) {
        if ((global phase == local phase) && (line == Valid))
                issue a refresh request for the line
}
release read and write requests
```

2.3.2 Processing Refresh Requests

The lower half of Fig. 2.7 shows the refresh processing logic that supports WB (n,m) and the other data-based policies that we consider. For each line, we have the line State bits and the Count bits. Note that the latter are not a counter but a set of bits. In our implementation, we use a 5-bit Count, which makes the Count and State overhead negligible.

The refresh request reaches the Decision Logic, which reads the State and

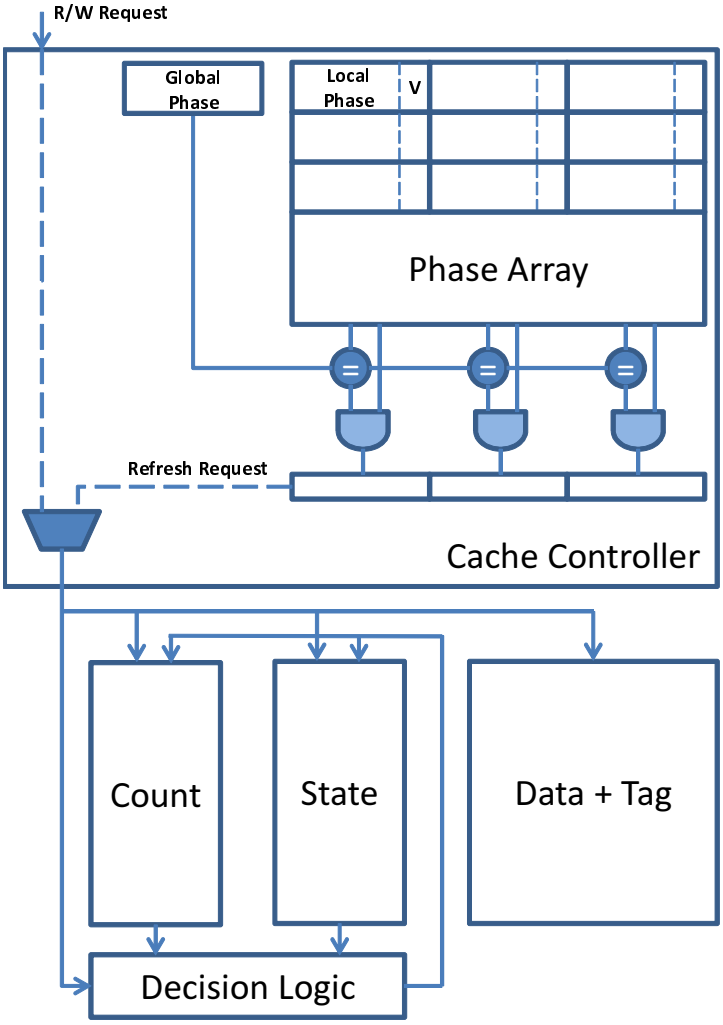


Figure 2.7: Phase array and logic to process refresh requests.

Count bits. Depending on the data policy (All, Valid, Dirty or WB(n,m)) and the value of the State and Count bits, the logic may change the Count bits, refresh the line, and write back or invalidate the line. Invalidation may trigger the sending of invalidations to upper-level caches, and this is initiated by the cache controller. In case of a WB(n,m) policy, the steps are shown below.

read Count
if (Count $>=$ **1)**
    refresh line, decrement Count
else if (State == Dirty)
    write back and change State to Valid Clean

/*the writeback automatically refreshes the line*/
Count = m
else if (State == Valid Clean)
invalidate
/*may also invalidate upper-level caches*/

2.3.3 Array Lookup and Update Energy

At the beginning of each phase, the cache controller scans the phase array to find lines which require a refresh. We quantify the energy cost of such a lookup. Considering the cache of 1 MB from the example above, we have to read 96 lines of 64 Bytes and perform 16,384 3-bit comparisons. Therefore, the energy overhead, E_{ovhd}, is $E_{ovhd} = 96 \times E_{array\ line\ access} + 16384 \times E_{3-bit\ comparator}$

From Synopsys synthesis tools, for a 32 nm process, we determine that $E_{array\ line\ access} = 20$ pJ and $E_{3-bit\ comparator} = 0.330$ fJ. Therefore, $E_{ovhd} = 1925.4$ pJ. From CACTI [18], the energy per access to a 1 MB cache for the same process is ≈ 100 pJ. Therefore, the energy overhead of the lookups is 19 cache accesses per phase. This is insignificant compared to the number of cache lines that are accessed in a phase.

The phase array is updated (read-modify-write) on every normal cache line read or write. As discussed above, the per access energy for a phase array line and for the cache are 20 pJ and 100 pJ, respectively. However, we will see in the evaluation that of all the energy consumed in the last level cache (which has 1 MB banks), dynamic energy accounts for very little. Hence, the overhead of updating the phase array is negligible as well. Overall, the small overhead of looking up and of updating the phase array pays off as substantial refresh energy savings.

2.4 Experimental Setup

We evaluate Refrint on a 16-core chip multiprocessor (CMP) system. Each core is a dual issue, out-of-order processor running the MIPS32 instruction set. Each core has a private instruction cache, a private data cache and a private second level (L2) cache. The 16 cores are connected through a 4x4 torus network. A shared third level (L3) cache is divided into 16 banks and each bank is connected to a vertex of the torus network. The addresses are statically mapped to the banks of the L3.

Table 2.2: Architectural and technology parameters and tools.

Architectural Parameters	
Chip	16 core CMP
Core	MIPS32, 2 issue out-of-order processor
Instruction L1	32 KB, 2 way. Access time (AT): 1 ns
Data L1	32 KB, 4 way, WT, private. AT: 1 ns
L2	256 KB, 8 way, WB, private. AT: 2 ns
L3	16 MB, 16 banks, WB, shared
L3 bank	1 MB, 8 way. AT: 4 ns
Line size	64 Bytes
DRAM	Round trip from controller: $\approx$80ns
Network	4 x 4 torus
Coherence	MESI directory protocol at L3
Technology Parameters	
Technology node	32 nm
Frequency	1000 MHz
Device type	LOP (Low operating power)
Temperature	330 Kelvin
Tools	
Architecture	SESC [19]
Timing & Power	McPAT [20] & CACTI [18]

Each L2 and each bank of L3 has dedicated logic to process refresh requests as shown in Fig. 2.7. We employ a MESI directory cache coherence protocol. The directory is maintained at L3. The architectural parameters are summarized in Table 2.2.

We model our architecture (cores, memory subsystem, and network) with the SESC [19] cycle-level simulator. The dynamic energy and leakage power numbers for cores and network are obtained from McPAT [20], while those for memories are obtained from CACTI [18]. Even though McPAT uses CACTI internally, it does not allow for an eDRAM memory hierarchy. Hence, we have to use CACTI as a standalone tool. We experiment with 2 different values of retention times, namely 50 μs and 100 μs. Barth et al. [21] report a retention time of 40 μs for eDRAM cells at 105 $^{\circ}$C. The retention time has an exponential dependence on temperature [22]. In this proposal, we target a low-voltage, low-frequency simple core and an energy-efficient architecture for which the temperatures are significantly lower than 105 $^{\circ}$C. Hence, we conduct experiments at the above mentioned retention times. Other experimental parameters like temperature and frequency

are also summarized in Table 2.2.

We compare a full-SRAM cache hierarchy (baseline) to one where L2 and L3 are eDRAM. To do a fair and simple comparison between the two, we have made a few simplifying decisions, which are listed in Table 2.3. Specifically, we assume that the L2 and L3 access times are the same in both hierarchies. We also assume that the access energies are the same. The leakage power of an eDRAM cell is 1/8th of that of an SRAM cell. In addition, for eDRAM, the time and energy to refresh a line is equal to the time and energy to access the line. Finally, a line can be refreshed in a cycle, when done in a pipelined fashion.

Table 2.3: Assumptions made for the memory cells in L2 and L3.

Parameter
eDRAM access time = SRAM access time
eDRAM access energy = SRAM access energy
eDRAM leakage power = 1/8 x SRAM leakage power
eDRAM line refresh time = eDRAM line access time
eDRAM line refresh energy = eDRAM line access energy

Table 2.4: Applications and input sizes run.

SPLASH-2		PARSEC	
FFT	2^{20}	Streamcluster	sim small
LU	512 x 512	Blackscholes	sim medium
Radix	2M keys	Fluidanimate	sim small
Cholesky	tk29.O		
Barnes	16 K particles		
FMM	16 K		
Radiosity	batch		
Raytrace	teapot		

We evaluate Refrint by running 16-threaded parallel applications from the SPLASH-2 [23] and PARSEC benchmark suites. The set of applications and the problem sizes are summarized in Table 2.4. In addition, we ran a synthetic application, called *fine share*, with fine-grained producer-consumer behavior. In this application, each thread independently reads and modifies a block of data and reaches a barrier. At the barrier, the block of data is passed to the adjacent thread. This is done in a loop. The time between consecutive barriers is less than the retention time, so that the frequency of accesses to the lines in the data block in the shared cache (L3) is higher than the refresh rate.

Each application is run at 2 retention times, 4 timing policies, 7 data policies and the baseline case, amounting to a total of 57 combinations. The parameter sweep is summarized in Table 2.5.

Table 2.5: Parameter sweep.

Retention time	50 μs, 100 μs	2
Timing policy	Periodic, Polyphase (num. phases = 1, 2, 4)	4
Data policy	All, Valid, Dirty, WB(4,4), WB(8,8), WB(16,16), WB(32,32)	7
Total combinations		57

2.5 Evaluation

In this section, we present our evaluation of Refrint. We present the effect of our policies on memory hierarchy energy (Section 2.5.1), total energy (Section 2.5.2) and execution time (Section 2.5.3).

Policies: We present results for *Periodic* and *Polyphase*. The *All* and *Valid* data-based policies do not create extra DRAM accesses. However, the *Dirty* and *WB(n,m)* (write back) policies create extra DRAM accesses by either discarding valid data or pushing dirty data to DRAM to save on-chip refresh energy. Therefore, to do a fair comparison, we take DRAM access energy [24] into account. In addition, we assume that at the end of the simulation all dirty data will be written back to main memory.

Applications: In Section 2.2.3 and in Fig. 2.6 we categorized applications into three classes based on data policies. In the course of our evaluation, we found that applications within a class responded similarly to our data policies. Table 2.6 shows the binning for our set of applications. In the following sections, rather than picking one representative application from each class, we will present average numbers for the entire class. To show the effectiveness of time policies we also present data from one synthetic application.

Table 2.6: Application binning.

Category	Applications
Class 1	FFT, FMM, Cholesky, Fluidanimate
Class 2	Barnes, LU, Radix, Radiosity
Class 3	Blackscholes, Streamcluster, Raytrace

2.5.1 Memory Hierarchy Energy

In Fig. 2.8, we have three bars. The first one shows the memory energy as the sum of L1, L2, L3 and DRAM energies, from bottom to top, normalized to the baseline memory hierarchy. The second and third bars show the L2 and L3 energies as the sum of their dynamic, leakage and refresh components, from bottom to top, normalized to the SRAM L2 and SRAM L3 energies, respectively. The data is for a naive eDRAM policy of refreshing all lines periodically at 50 μs, and have been averaged over all applications.

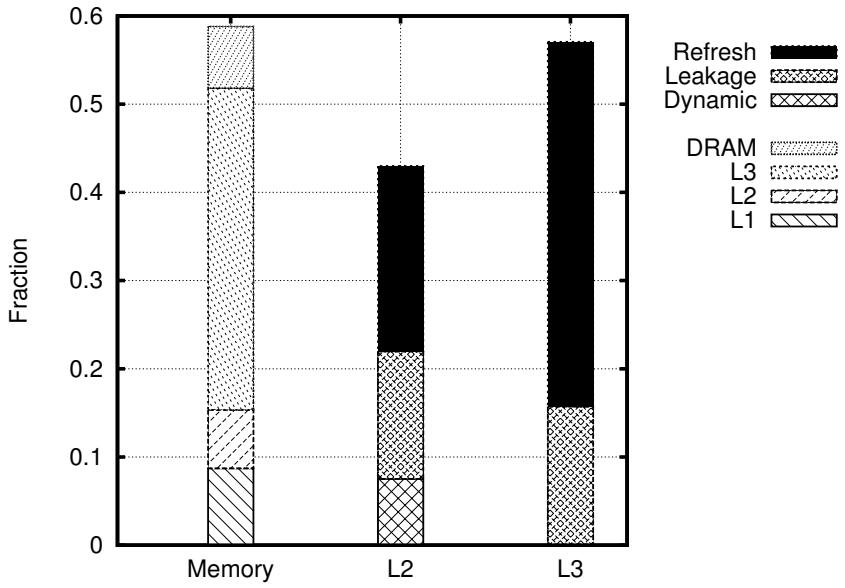


Figure 2.8: Memory system, L2 & L3 energy (normalized to the corresponding baseline energy).

We observe that L3 consumes the majority ($\sim$ 60%) of the total memory energy, of which about 70% is refresh energy. L2 consumes about 10% of the total memory energy, of which about 50% is refresh energy. The aim of our policies is to shave off as much refresh energy as possible. Next, we show the effectiveness of our polices in saving refresh energy in L3 and L2.

2.5.1.1 L3 Refresh Energy

To isolate the effectiveness of our policies in saving L3 refresh energy, we employ a Periodic timing policy and Valid data policy in L2. We do a parameter sweep on L3 refresh policies as summarized in Table 2.5.

In Fig. 2.9, we show the memory energy (averaged) for Class 1, Class 2 and Class 3 applications as the sum of on-chip dynamic, leakage, and refresh energies and DRAM energy. The fourth plot (labeled 'all') shows the average over all applications. On the X-axis, we have 2 sets of bars at retention times of 50 μs and 100 μs. Within each retention time, we have 4 time-based policies, namely Periodic (P) and Polyphase with 1, 2 and 4 phases (PP1, PP2, and PP4). For each time-based policy we have 7 data-based policies, namely *All, Valid, Dirty, WB(4,4), WB(8,8), WB(16,16)* and *WB(32,32)*. The bars are labeled as 'time-policy.data-policy', e.g., P.WB(4,4) stands for Periodic and WB(4,4) policy. The Y-axis represents total memory energy as the sum of on-chip dynamic, leakage, and refresh energies and DRAM energy, normalized to the baseline memory hierarchy energy. The policy P.all represents the naive eDRAM policy of refreshing all lines periodically.

In all classes of applications, the amount of dynamic energy remains almost the same across retention times and across policies because the amount of work done is the same. The amount of leakage energy varies because of the effect of the policies on execution time (Section 2.5.3). The main variation is in the amount of refresh energy, which is the focus of this proposal. The reduction in on-chip refresh energy (as a result of policies) comes at the cost of extra DRAM accesses, whose energy consumption has been taken into account.

Retention Time: As the retention time increases, the lines have to be refreshed less often and hence the amount of refresh energy reduces. The effect of the policies (timing and data) is most pronounced at smaller retention times.

Timing Policies: Polyphase policies (PP1, PP2, and PP4) always do better than Periodic because of two reasons. First, in Periodic, a cache cycle is spent for every line (valid or invalid) to determine if a refresh needs to be scheduled. In Polyphase, thanks to the phase array, this check is performed for multiple lines at once, and cache cycles are consumed only for issued refresh requests. Second, PP2 and PP4 can save refreshes compared to Periodic and PP1 by exploiting recently-accessed

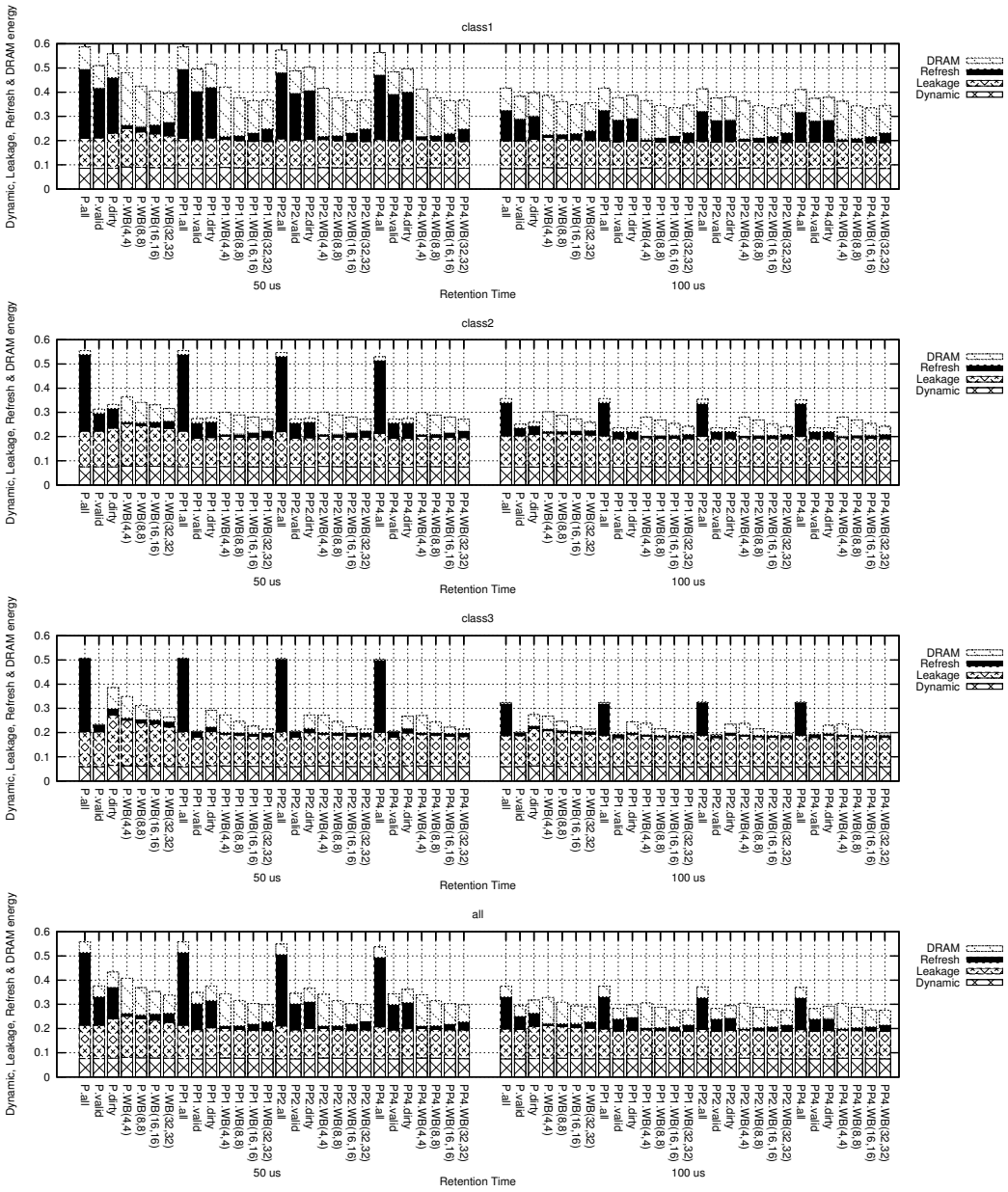


Figure 2.9: On-chip dynamic, leakage, refresh & DRAM energy (normalized to the baseline memory hierarchy energy).

lines. Unfortunately, for the applications considered, we do not see much benefit in increasing the number of phases in a retention period to PP2 or PP4. This is because the number of read/write accesses to L3 is very small in comparison to the number of refreshes. Hence, adding more phases is not noticeably effective at eliminating refreshes.

Data Policies: The effect of data policies is different in the three classes of applications. In Class 1 applications (large footprint, high visibility), *WB(n,m)* policies with relatively small values of (n,m) are effective at reducing the refresh component and the total memory energy in comparison to *All, Valid* and *Dirty* policies. In Class 2 applications (small footprint, high visibility), *WB(n,m)* policies are most effective with high values of (n,m). The *Valid* scheme does equally well for such applications. In Class 3 applications (small footprint, low visibility), any policy such as *Dirty* or *WB(n,m)* that attempts to reduce refresh energy pays a penalty in terms of leakage energy (due to increased execution time) or DRAM energy. The *Valid* scheme does best for this class of applications. Our observations are in line with our hypothesis from Section 2.2.3.

On average across all applications, Polyphase with 1 phase (PP1) does better than Periodic schemes. Also, the *WB(32,32)* policy does better than all other policies. At 50 μs, on average, the base refresh policy for eDRAM (Periodic All) consumes 56% of the energy of the conventional SRAM-based memory hierarchy. Our Polyphase WB(32,32) policy consumes only 30% of the energy of the conventional SRAM-based memory hierarchy.

2.5.1.2 L2 Refresh Energy

In the section above, we found that on average the Polyphase timing policy with 1 phase and the WB(32,32) data policy perform well for L3. Now, to isolate the effectiveness of our policies in saving L2 refresh energy, we freeze the L3 refresh policy to Polyphase with 1 phase and WB(32,32) and do a parameter sweep on L2 refresh policies. Since the L2 energy is a small fraction of the total memory energy, in Fig. 2.10 we show the impact of refresh policies only on L2 energy. Fig. 2.11 shows the impact of L2 policies on the total memory energy, averaged over all applications. In both plots, the X-axis is the same as in Section 2.5.1.1.

Retention Time: In both plots, as the retention time increases, the lines have to be refreshed less often and hence the amount of refresh energy reduces.

Timing Policies: Though not so significant, increasing the number of phases (keeping the data policy constant) shaves off about 5% refresh energy, as can be seen in Fig. 2.10.

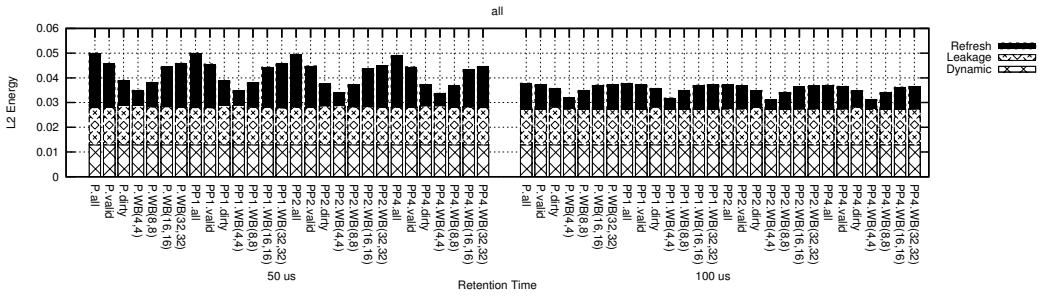


Figure 2.10: L2 dynamic, leakage and refresh energy (normalized to the baseline memory hierarchy energy).

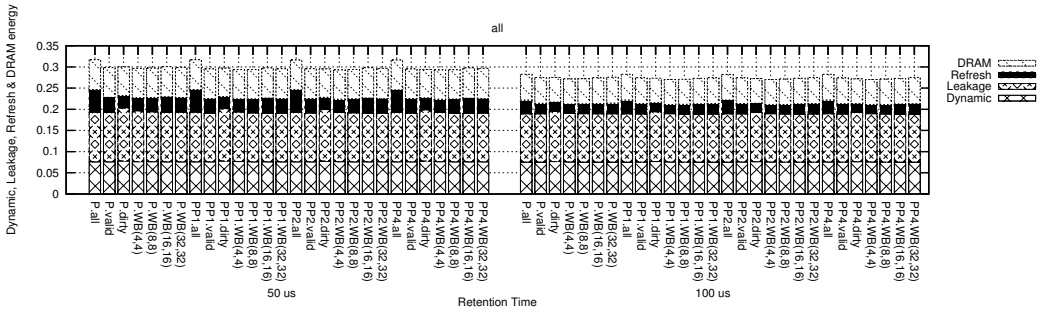


Figure 2.11: On chip dynamic, leakage, refresh & DRAM energy (normalized to the baseline memory hierarchy energy).

Data Policies: In Fig. 2.10, we notice that the WB(4,4) policy is highly effective at saving L2 energy. At 50 μs, on average across all applications, WB(4,4) saves 30% of the L2 energy compared to P.all. However, in Fig. 2.11, which shows the total memory hierarchy energy, we observe that the energy benefits of policies on L2 have leveled out. This is because the L2 refresh energy represents a small fraction of the total memory energy.

Therefore, to reduce implementation complexity, a periodic-valid refresh policy (P.valid) is the best choice for L2.

2.5.2 Total Energy

Using the result from Section 2.5.1.2 we freeze the L2 refresh policy to periodic-valid. In Fig. 2.12 we show the normalized total system energy (cores, caches, network and DRAM access energy) averaged over all applications, with a parameter sweep on L3 policies. The X-axis of the plots is the same as in Section 2.5.1.1. On average, across all applications, Polyphase with 1 phase and WB(32,32) still does the best. At 50 μs, on average, a system with the conventional refresh policy

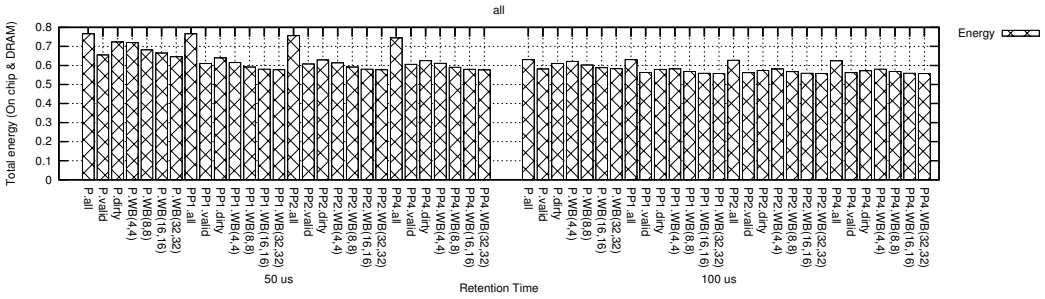


Figure 2.12: Total energy (normalized to the baseline system energy).

for eDRAM (Periodic All) consumes 77% of the energy of the baseline system. A system with our Polyphase WB(32,32) policy consumes 58% of the energy of the baseline system.

2.5.3 Execution Time

Using the same refresh policies and parameter sweep as in the above section, in Fig. 2.13 we show the normalized execution time averaged over all applications. The X-axis of the plot is the same as in Section 2.5.1.1. The Y-axis represents the total application execution time. All applications have the same trend across the retention times and policies. On average, with increasing retention times, the performance penalty reduces.

Timing Policies: Periodic schemes do worse in comparison to the Polyphase schemes. This is because periodic schemes consume a cache cycle per line, while Polyphase schemes only consume it if they issue a refresh request. We also see that, while Polyphase schemes with more phases per retention period could potentially save refreshes over those with fewer phases, our applications do not noticeably improve with more phases per retention period.

Data Policies: The Valid and WB(32,32) data policies are the best with respect to execution time, as they keep all the lines or keep them for a long enough time. The Dirty and other WB(n,m) policies incur a performance overhead, due to an increase in miss rates caused by extra invalidations and writebacks. The performance penalty for the WB(n,m) policy goes down as (n,m) grow. This is reasonable, as data is kept around for a longer time.

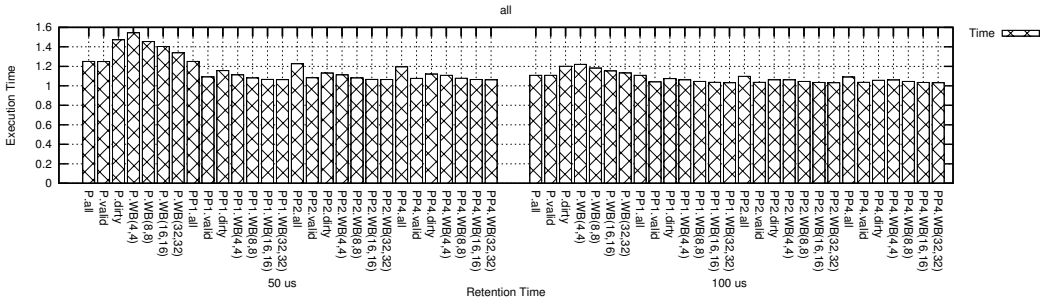


Figure 2.13: Execution time (normalized to the baseline execution time).

At 50 μs, on average, the execution time in a system with the base refresh policy for eDRAM (Periodic All) is about 25% longer than in the baseline system. The execution time in a system with our Polyphase WB(32,32) policy is only 6% longer than in the baseline system.

2.5.4 Effectiveness of Polyphase

As pointed out in Section 2.2.3.1 and observed above, applications from the SPLASH-2 and PARSEC suites do not benefit noticeably from time-based policies with phases. This is because such codes do not exhibit substantial amounts of fine-grained sharing, upper-level cache conflicts, or accesses to data uncacheable in upper-level caches. To show the effectiveness of Polyphase in one of these cases, we evaluate *fine share*, a microbenchmark with fine-grained sharing.

Fig. 2.14 shows the refresh energy in the shared L3 cache while running *fine share*. The X-axis is the same as in Section 2.5.1.1. The Y-axis is normalized to the L3 refresh energy for the Periodic Valid policy at 50 μs. The bars for the All policies extend beyond the range shown and are cut. From the figure, we see that, across the two retention times and all the data policies, there is a significant decrease in L3 refresh energy with an increase in the number of phases. For example, at 50 μs for data policy WB(32,32), the reduction from PP1 to PP4 is 50%.

2.6 Related Work

To reduce leakage power in conventional SRAM caches, several classes of approach have been proposed. One class includes Gated-V_{dd} [13] and Cache Decay

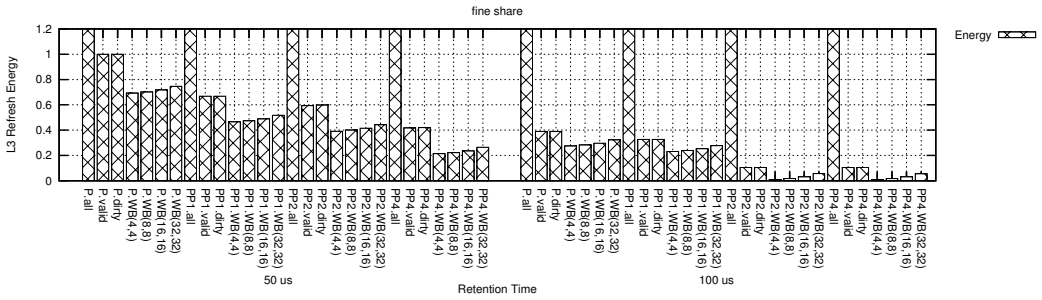


Figure 2.14: Effectiveness of Polyphase on a microbenchmark with fine-grained sharing.

[12, 16], which turn off cache lines that are not likely to be accessed in the near future, thereby saving leakage power. Cache Decay uses counters, which may have a significant cost for large lower-level caches. With this approach, the state of a cache line is lost when the line gets turned off.

A second approach is Drowsy Caches [11, 17]. Inactive lines are periodically moved to a low-leakage mode, where they cannot be read or written. This approach has the advantage that the data state is preserved, using two different supply voltage modes. This scheme will be less applicable in the future, where the difference between V_{dd} and V_{th} will be smaller.

Both of the previous approaches require design changes, power gating/voltage biasing circuitry, and have a non-negligible hardware overhead. On the other hand, eDRAM, the focus of our work, offers intrinsic leakage reduction and area benefits.

eDRAM cells can be logic-based or DRAM based. Logic-based eDRAM operates faster but is more expensive, as it complicates the logic manufacturing process. Logic-based eDRAM, as a feasible alternative to on-chip SRAMs, has been proposed in [25]. To make eDRAM delay characteristics closer to SRAM, and to simplify the process technology, Liang et al. [26] proposed the 3T-1D eDRAM cell for L1 caches. The proposed cell consists of three transistors and a diode which loses its charge over time, thereby requiring refresh. Hu et al. [27] proposed a dynamic cell with 4 transistors by removing two transistors that restore the charge loss in a conventional 6T SRAM cell. The 4T cell requires less area compared to the 6T SRAM cell, while achieving almost the same performance. However, the data access is slower and destructive, which can be solved by refreshing the data.

Hybrid memory cells have also been proposed to take advantage of the different

features that different memory cells offer. Valero et al. [28] introduced a macro-cell that combines SRAM and eDRAM at cell level. They implement an N-way set-associative cache with these macro-cells consisting of one SRAM cell, N-1 eDRAM cells, and a transistor that acts as a bridge to move data from the static cell to the dynamic ones. Although applicable to first-level caches, this approach is not effective for large shared caches, since the access patterns are not so predictable, and the data access characteristics at L1 caches do not hold true at lower-level caches.

In deep sub-micron technology nodes, when implementing an eDRAM-based on-chip cache, the power consumption and the performance overhead of refreshing the eDRAM cells become the main bottlenecks. An interesting approach is introduced by Venkatesan et al. [29]. It is a software-based mechanism that allocates blocks with longer retention time before allocating the ones with a shorter retention time. Using this technique, the refresh period of the whole cache is determined only by the portion of the cache used instead of the entire cache.

Ghosh et al. [30] proposed Smart Refresh, which is a technique to avoid unnecessary refreshes of lines in the main memory DRAM that have recently been read or written. To accomplish this, it uses timeout counters per line. Smart Refresh differs from Refrint in several ways. First, Refrint uses two approaches to eliminate refreshes: not refresh the lines that are not being used, and not refresh the lines that have recently been accessed. Smart Refresh is only relevant to the second approach. In addition, Smart Refresh needs to change DRAM arrays, which is harder to do, while Refrint, being on chip, can augment memory structures more easily. Finally, Smart Refresh has a counter per line, while Refrint just keeps count bits (no logic) per line. Of course, Smart Refresh lines are longer.

Using error-correction codes (ECCs) is another technique to reduce the refresh power [31, 3]. ECCs can tolerate some failures and hence allow an increase in the refresh time. By employing a stronger ECC, we can increase the refresh period and reduce the refresh power. Nonetheless, strong codes come with overheads in terms of storage, encoding/decoding power, area, and complexity.

2.7 Summary

To reduce the power consumed in the on-chip memory hierarchy of large many-cores, this proposal considered a low-leakage technology (eDRAM) and proposed

to refresh it intelligently for power savings. Our goal is to refresh only the data that will be used in the near future, and only if the data has not been recently accessed (and automatically refreshed). Our technique is called *Refrint*. We introduced the Refrint algorithms and microarchitecture.

We evaluated 16-threaded parallel applications running on a chip multiprocessor with a three-level on-chip cache hierarchy. Our results showed that Refrint is very effective. On average, an eDRAM-based memory hierarchy without Refrint consumed 56% of the energy of a conventional SRAM-based memory hierarchy. However, it increased the execution time of the applications by 25%. On the other hand, an eDRAM-based memory hierarchy with Refrint only consumed 30% of the energy of the conventional SRAM-based memory hierarchy. In addition, it only increased the execution time of the applications by 6%. In this environment, the contribution of refreshes in the energy remaining was negligible.

CHAPTER 3

MOSAIC

Since eDRAM refresh is an important problem, there is significant work trying to understand the characteristics of eDRAM charge retention (e.g., [32, 33, 8]). In reality, it is well known that different eDRAM cells can exhibit very different charge-retention properties and, therefore, have different refresh needs. However, current designs pessimistically assume worst-case retention times, and end up refreshing all the eDRAM cells in a module at the same, conservatively-high rate. This naive approach is wasteful.

Recent experimental work from IBM has shown that the retention time of an eDRAM cell strongly depends on the threshold voltage (V_t) of its access transistor [8]. We note that, since the values of V_t within a die have spatial correlation, then eDRAM retention times will also necessarily exhibit spatial correlation. This suggests that architectural mechanisms designed to exploit such correlation can easily save refresh energy.

Consequently, in this proposal, we first develop a new model of the retention times in large on-chip eDRAM modules. The model, called *Mosaic*, builds on process-variation concepts. It shows that the retention properties of cells in large eDRAM modules do exhibit spatial correlation. Then, based on the model, we develop a low-cost architectural mechanism to exploit such correlation and eliminate most of the refreshes.

Our architectural technique, also called Mosaic, consists of logically dividing an eDRAM module into logical regions or *tiles*, profiling the retention characteristics of each tile, and programming their refresh requirements in small counters in the cache controller. Such counters then trigger refreshes when it is time to refresh their corresponding tiles. With this architecture, we refresh each tile at a different rate.

Our results show that the Mosaic tiled architecture is both inexpensive and very effective. An eDRAM L3 cache augmented with Mosaic tiles increases its area by 2% and reduces the number of refreshes by 20 times. With Mosaic, we get very

close to the lower bound in refresh energy, and end up saving 43% of the total energy in the L3 cache.

This chapter is organized as follows: Section 3.1 discusses the background and motivation; Section 3.2 introduces the Mosaic model; Sections 3.3 and 3.4 present the Mosaic architecture; Sections 3.5 and 3.6 evaluate them; and Section 3.7 covers related work.

3.1 Background & Motivation

In this section, we discuss how eDRAM cells retain charge. We observe that the expected retention time and the one assumed in practice are off by orders of magnitude. We then present the distribution of the retention time and discuss its sources.

3.1.1 eDRAM Cell Retention Time

Fig. 3.1 shows an eDRAM cell. It consists of an access transistor and a storage capacitor. The logic state is stored as electrical charge in the capacitor. The capacitor loses charge over time through the access transistor — shown as I_{off} in the figure. Therefore, an eDRAM cell requires periodic refresh to maintain the correct logic state.

The leakage through the transistor depends on the threshold voltage (V_t) of the transistor. The higher the V_t, the lower the leakage and, therefore, the cell retains its logic value for longer. Conversely, a low V_t results in more leakage and, hence, the cell loses its logic value sooner. On the other hand, a higher V_t reduces the overdrive of the transistor and increases the access time of the cell. Therefore, there is a tradeoff between the cell access time and how long it retains its value.

We now derive a closed-form mathematical equation relating the parameters of the cell to its retention time. Let C be the storage capacitance, W and L the width and length of the access transistor, V the voltage applied to the gate of the access transistor, S_t the subthreshold slope (defined below), I_{off} the off drain current through the access transistor, and T_{ret} the retention time of the eDRAM cell. T_{ret} is defined as the time until the capacitor loses 6/10th of the stored charge [8], that is,

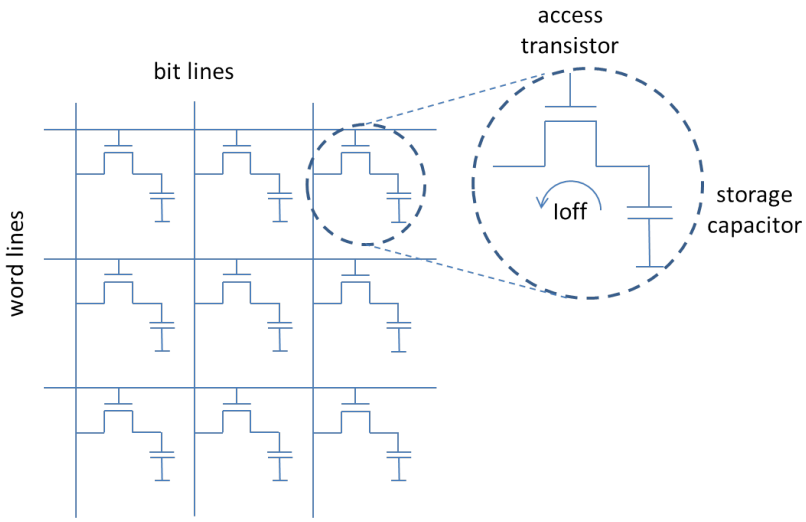


Figure 3.1: An eDRAM cell.

$$T_{ret} = \frac{0.6 \times C}{I_{off(V=0)}} \tag{3.1}$$

The definition of V_t is empirical. The definition varies from foundry to foundry, and across technology nodes. Kong et al. [8] define it as the gate voltage at which the current becomes the expression on the right in Eq. 3.2.

$$I_{off(V=V_t)} = 300 \times \frac{W}{L}\ nAmps \tag{3.2}$$

The subthreshold slope is defined as the inverse of the slope of the semi–logarithmic I_{off}-V curve, that is,

$$S_t = \frac{V_t - 0}{log_{10}(I_{off(V=V_t)}) - log_{10}(I_{off(V=0)})} \tag{3.3a}$$

$$= \frac{V_t}{log_{10}(\frac{I_{off(V=Vt)}}{I_{off(V=0)}})} \tag{3.3b}$$

Re-arranging and substituting,

$$I_{off(V=0)} = I_{off(V=Vt)} \times 10^{-V_t/S_t} \tag{3.4a}$$

$$= 300 \times \frac{W}{L} \times 10^{-V_t/S_t}\ nAmps \tag{3.4b}$$

Substituting Eq. 3.4b in Eq. 3.1 gives

$$T_{ret} = 0.6 \times C \times \frac{L}{W} \times 10^{V_t/S_t} \times 10^9/300\ sec \tag{3.5}$$

From [8], at 65nm technology, we get $C = 20\ fF$, $L = W = 100\ nm$, $V_t = 0.65\ V$ and $S_t = 112\ mV/dec$. Substituting these values in Eq. 3.5, we get $T_{ret} = 25.44\ ms$.

Therefore, we expect eDRAM cell retention times to be of the order of a few tens of milliseconds. However, in practice, eDRAM cells are refreshed with a period of the order of a few tens of *microseconds*. For example, Barth et al. [21] report a time of 40 μs. This is because manufacturing process variations result in a distribution of retention times and, to attain a high yield, manufacturers choose the retention time for the entire memory module to be the one of the leakiest cells.

3.1.2 Retention Time Variation

It is well known that there is variation in the retention time of eDRAM and DRAM cells (e.g., [32, 33, 8]). The overall distribution and the sources of variation have also been identified. Fig. 3.2 shows a typical eDRAM retention time distribution [8]. The X axis is $log_{10}\ T_{ret}$. The Y-axis is the cumulative density function of the number of cells under a given retention time. The Y axis uses a normal distribution scale — that is, 0 represents the fraction 0.500, -1σ represents the fraction 0.158, and so on as shown in Table 3.1.

The figure shows that the retention time distribution has two components, namely the *Bulk Distribution* and the *Defect Tail Distribution*. The Bulk Distribution includes the majority of cells. Specifically, since the figure shows that the Bulk Distribution goes from approx. -4σ to ∞, it includes the 0.999968 fraction of the cells — as given by the area under the curve of a normal distribution from -4σ to ∞. In addition, the fact that it appears as a straight line in the log-normal plot of Fig. 3.2 indicates that $log_{10}\ T_{ret}$ follows a normal distribution for the Bulk — or that T_{ret} follows a log-normal one for the Bulk.

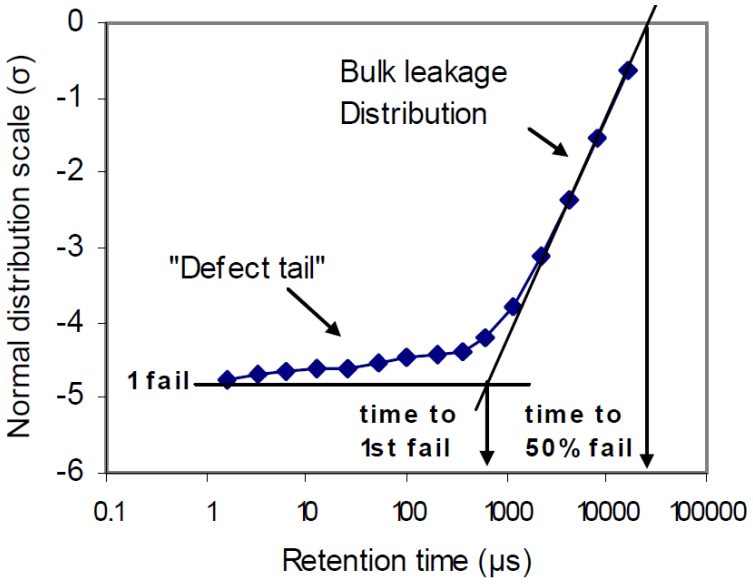


Figure 3.2: Typical eDRAM retention time distribution [8].

Table 3.1: Area under the curve for a normal distribution.

Sigma (σ)	Fraction
0.0	0.500000
-1.0	0.158655
-2.0	0.022750
-3.0	0.001349
-4.0	0.000031
-4.5	0.000003

Based on experimental data, Kong et al. [8] from IBM say "We demonstrate that the T_{ret} (Bulk) Distribution can be attributed to array (i.e., access transistor) V_t variation". This is a *key observation*, and is consistent with what we know about V_t's process variation distribution. Indeed, it is accepted that process variation in V_t follows a normal distribution [8]. If we take the log of Eq. 3.5, we obtain

$$log_{10}\, T_{ret} = \frac{V_t}{S_t} + expression \tag{3.6}$$

which shows that a normal distribution of V_t results in a normal distribution of $log_{10}\ T_{ret}$ and, hence, a log-normal distribution of T_{ret}. This agrees with the straight line in Fig. 3.2.

The Tail Distribution includes very few cells. Since it covers the area under the curve from $-\infty$ to approx. -4σ in Fig. 3.2, it includes only the 0.000031 fraction of the cells, or 31 ppm (parts per million). The fact that it appears as a straight line in Fig. 3.2 indicates that $log_{10}\ T_{ret}$ follows a normal distribution for the Tail — hence, T_{ret} follows a log-normal one for the Tail.

The source of the T_{ret} Tail Distribution has been attributed to random manufacturing defects. These defects manifest themselves as leaky cells. However, not all the cells following the Tail Distribution are considered defective. Only the cells in the region $-\infty$ to -4.5σ (about 3 ppm) are considered defective and are handled by redundant lines provided by ordinary designs [34, 8].

In the distribution above, the -4.5σ point represents a retention time of 45 μs. Barth et al. [21] have reported retention times of 40 μs as well. Therefore, it is clear that, overall, eDRAMs are refreshed at a very pessimistic rate. Since process variation in the V_t of the access transistor governs the distribution of almost all the cells, we look at it in more detail next.

3.1.3 Process Variation in the Threshold Voltage

Process variation in the V_t has two components, namely, *systematic* and *random*. Systematic variation is introduced by lithographic tools such as steppers, and exhibits high spatial correlation [35, 36, 37] — i.e., nearby transistors have similar V_t. Random variation is the result of material defects, dopant fluctuation, and line edge roughness, and is essentially white noise. The total variation is a superposition of the systematic and random components.

VARIUS [38] and other variation-modeling tools model the two components with normal distributions. Each distribution has its own sigma, namely, σ_{sys} and σ_{rand}. The superposition of both components results in an overall normal distribution for V_t's variation, with a sigma σ_{tot} equal to $\sqrt{\sigma_{sys}^2 + \sigma_{rand}^2}$. It is the *combined systematic and random components* of V_t's variation that induce the Bulk Distribution in Fig. 3.2.

From Eq. 3.6, we observe that the spatial correlation in V_t will result in spatial correlation in the retention times of eDRAM cells — i.e., eDRAM cells that are spatially close to each other will have similar retention time values. In this proposal, we exploit this property to eliminate most refreshes.

3.2 The Mosaic Retention Time Model

We want to develop a new model of eDRAM retention time that can help us to understand and optimize eDRAM refreshing. This section describes our model, which we call *Mosaic*.

3.2.1 Extracting the Values of Retention Parameters

To build the model, we first need to obtain the values for the key parameters of the T_{ret} Bulk and Tail Distributions in Fig. 3.2. Specifically, we need: (i) the mean and sigma of the Bulk Distribution $(\mu_{Bulk}, \sigma_{Bulk})$, (ii) the mean and sigma of the Tail Distribution $(\mu_{Tail}, \sigma_{Tail})$, and (iii) the fraction of cells that follow the Tail Distribution (ρ). From Kong et al. [8], we obtain that $\mu(V_t) = 0.65\ V$, $\sigma(V_t) = 0.042\ V$, and $S_t = 112\ mV/dec$. Therefore, from Eq. 3.6, and computing *expression* based on Eq. 3.5, we get the parameter values for the Bulk Distribution:

$$\mu_{Bulk}(log_{10}\ T_{ret}) = \frac{\mu(V_t)}{S_t} - 7.40 = -1.594$$
$$\sigma_{Bulk}(log_{10}\ T_{ret}) = \frac{\sigma(V_t)}{S_t} = 0.375$$

Kim and Lee [33] observe that the peak of the Bulk Distribution and the peak of the Tail Distribution in DRAMs are off by approximately one order of magnitude. This is 1 in log_{10} scale, which is approximately 3 times the value that we have just obtained for σ_{Bulk}. Hence, in our model, we estimate the peak of the Tail Distribution ($\mu_{Tail}(log_{10}\ T_{ret})$) to be $3 \times \sigma_{Bulk}(log_{10}\ T_{ret})$ to the left of the peak of the Bulk Distribution ($\mu_{Bulk}(log_{10}\ T_{ret})$):

$$\begin{aligned}\mu_{Tail}(log_{10}\ T_{ret}) - \mu_{Bulk}(log_{10}\ T_{ret}) &= -3 \times \sigma_{Bulk}(log_{10}\ T_{ret}) \\ &= -1.125 \\ hence, \quad \mu_{Tail}(log_{10}\ T_{ret}) &= -2.719\end{aligned}$$

We obtain the last two parameters, namely $\sigma_{Tail}(log_{10}\ T_{ret})$ and ρ, by curve-fitting the data in Fig. 3.2. We obtain

$$\sigma_{Tail}(log_{10}\ T_{ret}) = 1.8$$
$$\rho = 0.000020\ (20\ ppm)$$

This value of ρ is more accurate than our initial estimation of 31 ppm for the Tail

Distribution in Sec. 3.1.2. The final parameter values are summarized in Table 3.2. With these values, we generate the curve shown in Fig. 3.3. On this same graph, we superpose the experimental data from Fig. 3.2 in circles. We can see that the curve fits the experimental data. Moreover, our parameters are in agreement with an observation made in [32] that the σ of the Tail and the Bulk Distributions have a ratio of 4.87. This ratio for Mosaic is 4.80. Finally, in this curve, the -4.5σ point (3 ppm) corresponds to a retention time of about 45 μs.

Table 3.2: Parameter values extracted from the data in [8].

Parameter	Value
$\mu_{Bulk}(log_{10}\ T_{ret})$	-1.594
$\sigma_{Bulk}(log_{10}\ T_{ret})$	0.375
$\mu_{Tail}(log_{10}\ T_{ret})$	-2.719
$\sigma_{Tail}(log_{10}\ T_{ret})$	1.8
ρ	20 ppm

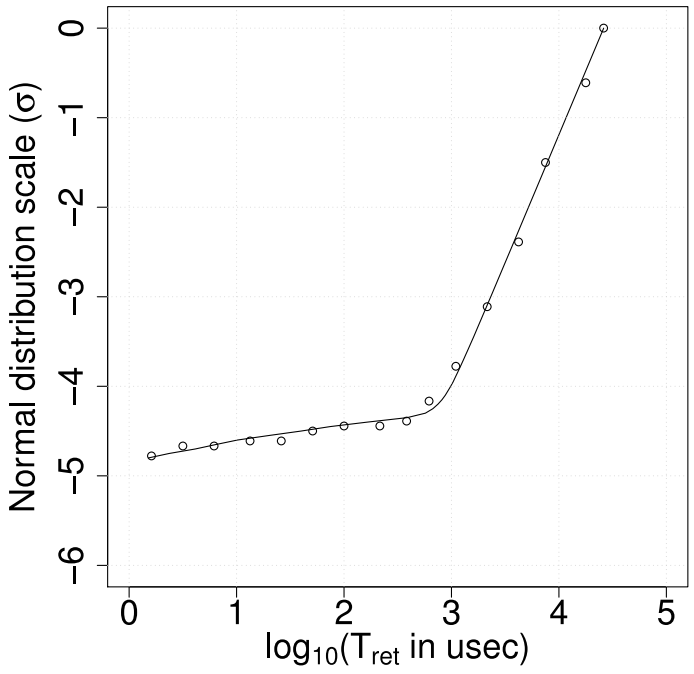


Figure 3.3: Retention time curve generated by Mosaic compared to the experimental data in [8].

3.2.2 Generating a Spatial Map of Retention Times

The parameter values for the Bulk and Tail Distributions extracted in the previous section are not enough to generate a spatial map of the T_{ret} values in a given memory module. The reason is that the $\sigma_{Bulk}(log_{10}\ T_{ret})$ has a random and a systematic component, and the latter has a spatial correlation function. These effects are caused by the V_t distribution, as per Eq. 3.6. Different values for the σ break down into systematic and random, and the spatial correlation does not change the Bulk line in Fig. 3.2, as long as the total σ stays constant. However, the values produce very different spatial maps of T_{ret}. For example, if the fraction of σ coming from its systematic component is high and the correlation distance is long, there will be spatial clusters of eDRAM cells with similar T_{ret}. For the Tail Distribution, since it does not have any systematic component, we do not need any more information.

Overall, to generate a spatial map of the T_{ret} in an eDRAM memory module, we need to know: (i) the values in Table 3.2, (ii) the breakdown of $\sigma_{Bulk}(log_{10}\ T_{ret})$ into random and systematic components, and (iii) the correlation function for the systematic component. Our observation is that we can obtain (ii) and (iii) based on published data on the *V_t variation of the access transistors*. Specifically, as per Eq. 3.6, the breakdown of $\sigma_{Bulk}(log_{10}\ T_{ret})$ into random and systematic components is the same as the breakdown of $\sigma(V_t)$. Similarly, the correlation of $log_{10}\ T_{ret}$'s systematic component follows the correlation of V_t's systematic component.

3.2.3 Procedure to Generate a Spatial T_{ret} Map

Let us now generate a spatial map of T_{ret} values for an on-chip eDRAM memory module of 1024×1024 cells that distribute according to the parameter values of Table 3.2. For the V_t variation parameters of the access transistors, we will assume the following. First, following Karnik et al. [39], the $\sigma(V_t)$ has equal components of systematic and random components — i.e., $\sigma_{sys} = \sigma_{rand} = \sigma/\sqrt{2}$. Secondly, the correlation function for V_t's systematic component follows the Spherical function described in VARIUS [38] with a correlation distance ϕ of 0.4.

To generate the spatial T_{ret} map, we first generate the one for its Bulk Distribution and then superpose the one for its Tail Distribution. For the Bulk one, we first generate the spatial map for the systematic component and then superpose the

one for the random component. To generate the T_{ret}'s Bulk Distribution maps, we will first proceed with the intermediate step of generating the V_t maps and then use Equation 3.6 to generate T_{ret}'s Bulk maps. This is done for pedagogical reasons, since we could instead directly generate T_{ret}'s Bulk maps using (i) the μ_{Bulk} and σ_{Bulk} of Table 3.2, (ii) the breakdown of $\sigma(V_t)$, and (iii) the correlation of V_t's systematic component.

3.2.3.1 Spatial Map for $log_{10}\ T_{ret}$'s Bulk Distribution

Following the VARIUS methodology [38], we lay out an imaginary grid of $N_x \times N_y$ points on top of the eDRAM memory module. We then invoke VARIUS with $\mu(V_t)$ = 0.65 V, $\sigma(V_t)$ = 0.042 V (these two values are from Kong et al. [8]), and correlation distance ϕ = 0.4 (this is one of our assumptions). We obtain a spatial map of V_t's systematic component, as shown in Fig. 3.4a. We then obtain $N_x \times N_y$ samples of a normal distribution with $\mu = 0$ and $\sigma_{rand}(V_t)$, without correlation, as shown in Fig. 3.4b. As expected, the spatial map looks like white noise. We then superpose both maps, point per point, and obtain the total V_t map in Fig. 3.4c. Finally, the spatial map for $log_{10}\ T_{ret}$'s Bulk Distribution is obtained from the spatial map of the total V_t by applying Eq. 3.6 to every point in the grid. The resulting map is shown in Fig. 3.4d.

3.2.3.2 Spatial Map for $log_{10}\ T_{ret}$'s Tail Distribution

We use a normal distribution with the μ_{Tail} and σ_{Tail} of Table 3.2, and take $N_x \times N_y \times \rho$ samples with no correlation. We place them randomly on the $N_x \times N_y$ grid. The locations of the Tail cells are marked by circles in Fig. 3.4e.

3.2.3.3 Spatial Map for Overall $log_{10}\ T_{ret}$'s Distribution

We obtain the spatial map for the overall $log_{10}\ T_{ret}$ by replacing points in the Bulk Distribution with those in the Tail Distribution. The result is Fig. 3.4f, where the locations of the Tail cells are marked in circles. The map is almost the same as the one with only Bulk, since only a fraction ρ is replaced by Tail cells.

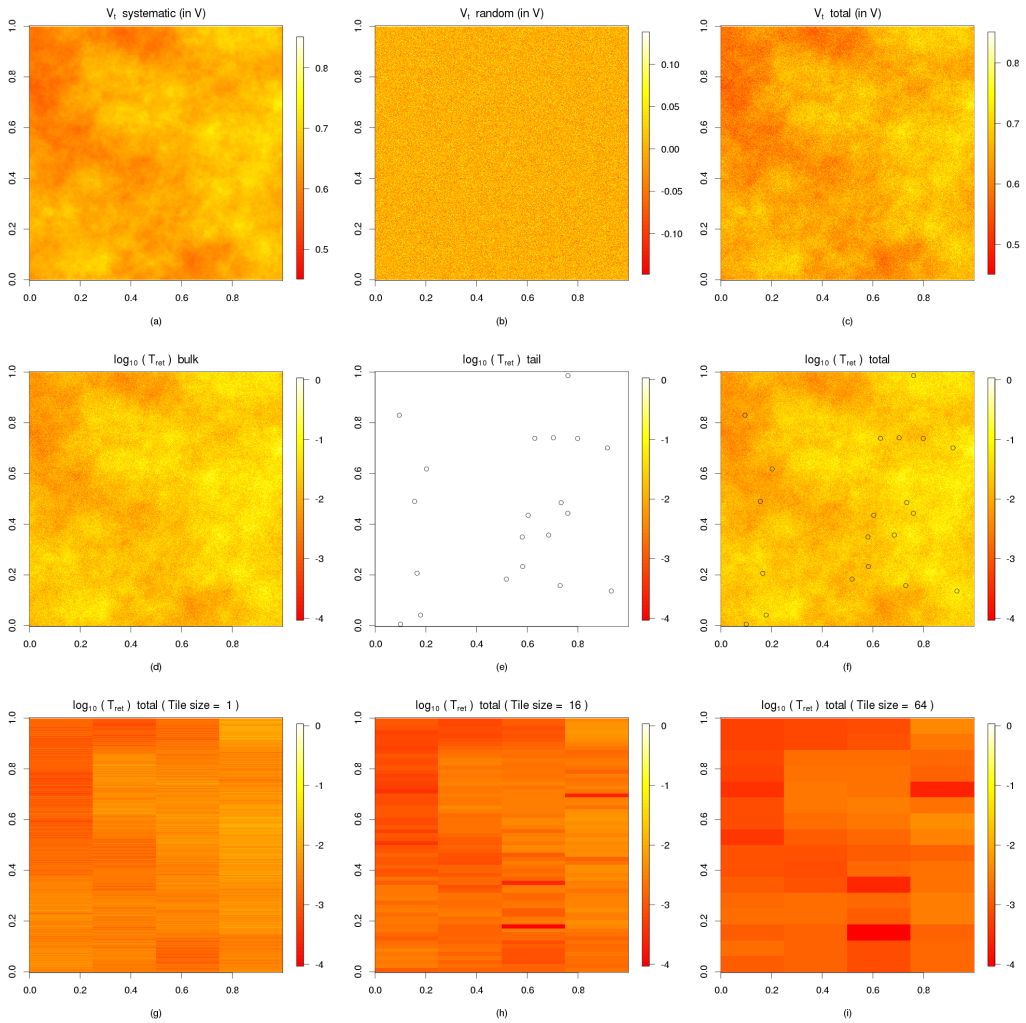


Figure 3.4: Various spatial maps of V_t and $log_{10}\ T_{ret}$ for an on-chip eDRAM memory module.

3.2.4 Using the Mosaic Retention Model Across Generations

As we move from one eDRAM generation to the next, we need to recompute the new distribution of T_{ret}. The Bulk Distribution of T_{ret} can be easily computed from the distribution of the access transistor's V_t — obtained with V_t's μ, σ, σ breakdown, and ϕ. This information on V_t for the new eDRAM generation may be easily available from the manufacturing process. The Tail Distribution of T_{ret}, however, needs to be characterized with direct measurements that extract $\mu_{Tail}(log_{10}\ T_{ret})$, $\sigma_{Tail}(log_{10}\ T_{ret})$, and ρ.

In some cases, it may be too expensive or difficult to characterize the Tail Dis-

tribution of T_{ret} for a new generation. In these cases, it may be possible to reuse the known Tail Distribution from the older eDRAM generation (if it has changed relatively little), and combine it with the V_t information of the new eDRAM generation, to generate the distribution of T_{ret} for the new eDRAM generation.

3.3 The Mosaic Architecture

3.3.1 Insight and Opportunity

The analysis in the previous section has provided a useful insight: since the retention time of an eDRAM cell is highly dependent on its access transistor's V_t, and V_t has well-known spatial correlation properties, then the retention time also has spatial correlation. This fact offers an opportunity to save refresh energy. Specifically, we can logically group cells into regions, profile their retention time, and set up time counters to refresh the regions only at the frequency that each one requires. With reasonable spatial correlation, the hardware cost of the counters will be minimal.

As an example, consider the eDRAM of Fig. 3.4f. Let us organize it as a 4-way set associative memory with 256-bit lines. Since eDRAM reads, writes and refreshes are performed at line granularity, we need the *line-level* distribution of $log_{10}\, T_{ret}$. For this, we set a line's T_{ret} to the minimum of the T_{ret} of the bit cells constituting the line. The result is shown in Fig. 3.4g.

If $Nlines$ is the number of lines in the memory module, the per-line retention time map gives an absolute lower bound of the number of refreshes required to prevent data loss:

$$Min.\, refreshes/sec = \sum_{i=1}^{Nlines} \frac{1}{T_{ret\_line\_i}} \tag{3.7}$$

This lower bound would be hard to attain. First, the memory has a limited number of ports, and only a number of lines equal to the number of ports can be refreshed simultaneously. Hence, some of the refreshes may need to be delayed. As a result, to ensure correctness when multiple lines need to be refreshed at the same time, and some refreshes need to be delayed, we need to provide a timing guardband.

In addition, providing a counter per line is too expensive. Kaxiras et al. [12]

have estimated the cost of an N-bit counter to be $40N + 20$ transistors. Therefore, even a 2-bit counter per 256-bit line would amount to a 40% area overhead. Solutions using Bloom filters lose accuracy.

The striations seen in Fig. 3.4g indicate that there is spatial correlation in the T_{ret} of adjacent lines — especially within the same way. Hence, we group sets of contiguous lines from the same way into *Tiles*, setting T_{ret} for the tile to the minimum of the T_{ret} of the lines constituting the tile. Figs. 3.4h and 3.4i show the tile-level distribution of $log_{10}\ T_{ret}$ for tile sizes of 16 and 64 lines, respectively. We can see that there are regions of spatial locality. The resulting distribution looks like a mosaic of tiles. With this design, the hardware cost of a counter is amortized over a whole tile. However, we have to refresh the whole tile whenever the counter rolls down to zero.

3.3.2 Profiling the Retention Times

Mosaic needs to profile the retention times of the tiles, for example at boot time. Literature from industry such as IBM [2, 8], Intel [3], Samsung [33], and Toshiba [32], and academic groups [22, 40, 29] have proposed and/or used T_{ret} profiling schemes. However, recent work [41] has pointed out that T_{ret} profiling has to deal with Data Pattern Dependence (DPD) and Variable Retention Time (VRT) effects. As suggested in [41] and [2], profiling in the presence of DPD can be best done by using a variety of manufacturer-provided test patterns — e.g., all 0s/1s, checkerboard, walk and random. One of the papers [41] also points out that VRT changes are slow (in the order of hours and sometimes a day) and, therefore, one could profile periodically. Note that a tile in Mosaic will include over 10,000 cells. With many cells, it is possible that, macroscopically, these effects exhibit relatively less external variation across measurements. In reality, sophisticated profiling techniques are still a subject of research in this area, and paramount to dynamic-memory manufacturers. Once the tiles are profiled at boot time, the per-tile T_{ret} count in cycles is stored in a small SRAM in the cache controller.

3.3.3 Temperature Adaptation

The V_t of a transistor is a function of temperature (*T*) [42]. Therefore, the access transistor's leakage current (Eq. 3.4b) and T_{ret} (Eq. 3.5) vary strongly with *T*.

Empirical data [43, 22] suggests the following exponential relationship between T_{ret} and *T*: there is approximately a $1.3x$ reduction in T_{ret} for every 10 °C increase in *T*. Specifically, let T_1 and T_2 be the *T* in °C and T_{ret_1} and T_{ret_2} their corresponding T_{ret}, then

$$T_{ret\_2} = T_{ret\_1} \times e^{-0.0268 \times (T_2 - T_1)} \tag{3.8}$$

The boot-time profiling temperature (T_0) and the corresponding T_{ret_0} values are stored in the SRAM. At run time, a thermal sensor measures *T*. If the *T* is T', then when Mosaic reads the SRAM, it computes the new T'_{ret} as $f(T_0, T_{ret\_0}, T' + Guardband)$, using Eqn. 3.8. We add a small *T* guardband as a safeguard against *T* changes between consecutive refreshes of the tile. This guardband is only a few degrees, as the time between refreshes is at most a few ms, and *T* changes slowly.

For these computations, we need a lookup table (LUT) and an 8-bit adder and multiplier. The exponential portion of Eqn. 3.8 is stored in a 32-entry LUT (from -40 °C to 120 °C in 5 °C steps). Each LUT entry is 8 bits.

In an advanced design, we may want to consider a per-tile T, rather than a single T for the whole eDRAM module. In this case, we would need to store a per-tile reference temperature T_0 in addition to a per-tile T_{ret_0}. At run time, *T* sensors would provide the T' of each tile. Then, for each tile, the algorithm would use the local T' and the local T_0 to apply the correction to the local T_{ret_0}.

3.3.4 Designing the Refresh Counters

Assuming that we have accurately profiled each tile's T_{ret}, we now consider the design of the refresh counters. A refresh operation is akin to a read or a write access to the cache line. However, the refresh operation takes precedence over normal accesses. In addition, we assume that a refresh operation takes one cycle when done in a pipelined fashion. If a memory module is organized into banks, then the banks can be refreshed in parallel. In a given bank, the number of ports sets the number of refreshes that can proceed in parallel; if the bank has a single port, then only one refresh can be done in the bank per cycle.

In this proposal, we assume one port per bank. Therefore, the minimum number of cycles required to refresh a whole bank is $Nlines$, which is the number of lines per tile times the number of tiles per bank. In the worst case, all the lines of the bank might require a refresh at the same time. To handle this corner case, we need

to use a time guardband equal to the maximum time between requesting a refresh and being serviced. This guardband is equal to the time required to refresh all the lines of the bank, that is,

$$Guardband = \frac{Nlines}{f} \tag{3.9}$$

where *f* is the frequency of the cache. Therefore, the corrected value of the retention time to be used for a tile, T'_{ret}, is

$$T'_{ret} = T_{ret} - Guardband \tag{3.10}$$

A second correction comes from the fact that Mosaic uses counters to track time. A counter increments in units of *Step*. Therefore, T'_{ret} has to be rounded to the highest multiple of Step that is less than T'_{ret}. Combining guardbanding and rounding off, the value of the retention time to be used, T''_{ret}, is

$$T''_{ret} = n \times step \mid n \times step \leq T'_{ret} < (n+1) \times step \tag{3.11}$$

3.3.5 Overall Mosaic Design & Operation

Fig. 3.5 shows the hardware needed by Mosaic to refresh a cache bank. The cache controller for the bank has a programmable clock divider, a down-counter for each of the tiles in the bank, a sequencer, an LUT with an adder and multiplier, and a small Retention Profile SRAM. The latter stores data obtained from profiling during boot time: T_{ret} for each of the tiles in the bank, and the bank's temperature (T_0). The LUT with the adder and multiplier are used for *T* adaptation (Section 3.3.3).

The programmable clock divider takes a reference clock as input and creates a clock pulse of Step, which is typically the -4.5σ point, i.e., about $45\ \mu s$. This signal acts as the clock for all the per-tile counters. Each counter is an N-bit down-counter that tracks the retention time of one tile. All counters are initialized to the values in the SRAM followed by the adjustments due to *T* (Section 3.3.3) and other corrections (Section 3.3.4).

At every Step, the sequencer rolls down all the counters one by one. For a given counter, it first decrements it and then compares its value to zero. If it is zero, the sequencer schedules refreshes for all the lines in the corresponding tile. Next, it

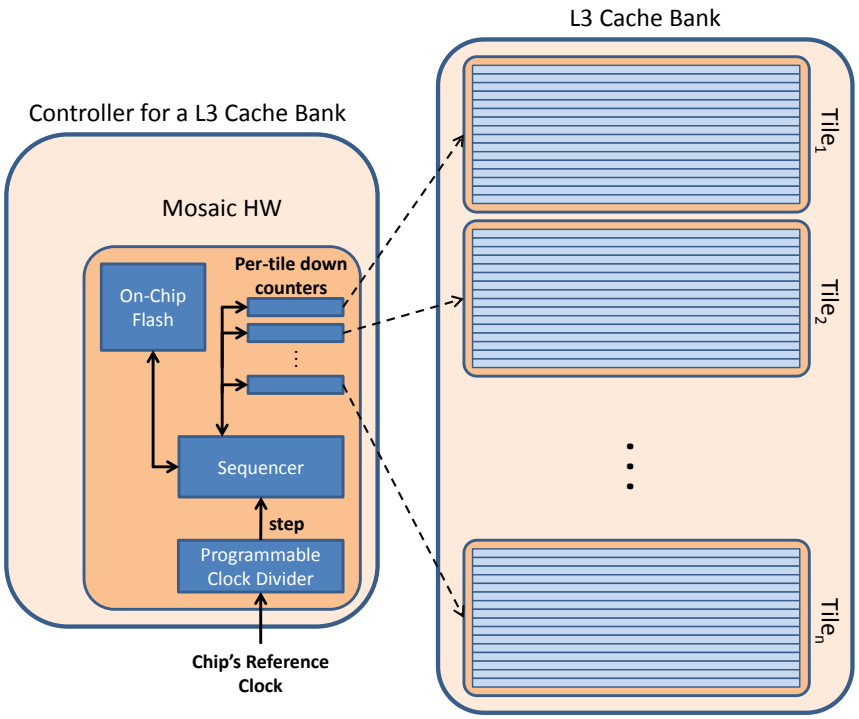


Figure 3.5: Hardware required by Mosaic to refresh a cache bank.

reloads the value of the counter after reading the SRAM and adjusting it for *T* and the other corrections. Then, the sequencer moves to the next tile. The process continues until all the counters get decremented.

3.3.6 Area and Energy Overheads

Mosaic induces little area and energy overhead. To see why, assume that Fig. 3.5 corresponds to a 1 MB cache bank with 64-byte lines and 16-line tiles. Such an organization has 1024 tiles in the bank. We can use down-counters with 8 bits which, if they step at 45 μs, can cover a T_{ret} of nearly 12 ms.

Consider first the area and energy overhead of the Retention Profile SRAM. Its storage capacity is about 1 KB, which is 0.1% of the data array in the cache bank. Hence, it takes an insignificant area. The SRAM is accessed only when a counter rolls down to zero. Given the T_{ret} distribution, this happens infrequently — typically, every few μs, since we have 1024 tiles and typical T_{ret} values are equal to a few ms. Hence, the overall dynamic energy associated with reading the

SRAM is insignificant. The same is the case for its leakage power.

Next, consider the area and energy overhead of the counters. If we assume that an N-bit counter needs about $40N + 20$ transistors [12], then adding an 8-bit counter per each 16-line tile adds 4% more transistors. In addition, since the counters are only accessed every 45 μs, their dynamic energy contribution is very small. Similarly, their leakage power is negligible. In Section 3.6, we estimate their area and energy overhead accurately using the Synopsys Design Compiler.

Finally, consider the 32-byte LUT and the 8-bit adder and multiplier. Using McPAT [20], we estimate that their area is 0.2% of the area of the cache bank, which is negligible. Also, the energy overhead of these structures is very small, as they are accessed only every few μs as the SRAM.

3.4 Discussion

We can approach the lower bound of refresh energy (Eq. 3.7) if we can afford per-line counters with an arbitrary number of bits and Step sizes. However, this solution is very expensive in both area and energy. On the other hand, we can avoid the area and energy overheads altogether if we refresh the entire eDRAM module at a constant rate, as it is currently done. There is a clear tradeoff between the potential refresh-energy savings and the overheads we are willing to incur.

As the number of lines per tile increases, the area and energy overheads of the counters decrease. However, the lines constituting the tile are refreshed at the rate needed by the line with the lowest retention in the tile. Therefore, we perform more refreshes than required and move away from the lower bound.

As the number of bits in the counter increases, the area and energy overhead increases, but in return we create more bins and therefore reduce the number of refreshes. The benefits saturate as soon as the range of the counter approaches the higher retention times of the distribution.

We do not experiment with counter Step sizes. A Step size larger than the -4.5σ point, such as 100 μs, would require disabling lines with retention times in the range 45 - 100 μs. It would allow us to investigate solutions that trade-off cache capacity for refresh energy gains. A Step size less than the -4.5σ point would increase the granularity of the counter, but decrease the range of the counter. Initial results suggested that, given the same number of bits, the benefits of a larger range with coarser granularity outweigh those of a smaller range with finer granularity.

We can potentially attain higher energy savings by using variable-sized tiles. For example, if there is a line or a small region of lines that have very different T_{ret} than their neighbors, we can save refreshes by placing them in a tile of their own. In contrast, if there is a very large region of lines that have similar T_{ret}, then we can save counter overhead and energy by placing all of them in a giant tile. However, while having variable-sized tiles may be attractive, it has hardware reconfiguration costs and design complexity. Further, as we will see in Section 3.6, Mosaic with fixed-sized tiles already eliminates the majority of refresh energy at a very modest hardware cost. Hence, there is little room to improve.

Finally, given an eDRAM module, the fraction of refresh power that Mosaic eliminates does not depend on the application running or its input data size. Mosaic's refresh savings depend on the T_{ret} variation profile, and the hardware parameters of the eDRAM module and Mosaic.

3.5 Evaluation Setup

We evaluate Mosaic with simulations of a chip multiprocessor (CMP) with 16 cores. Each core is a dual-issue, out-of-order engine. The cache hierarchy consists of per-core private L1 instruction and data caches, a per-core private L2 cache, and a shared L3 cache. The L3 is divided into 16 banks. Each bank is close to one core and has a statically-mapped range of addresses. A 4x4 torus on-chip-network connects the 16 nodes. The CMP uses a MESI directory coherence protocol maintained at the L3. The architectural parameters are shown in Table 3.3.

The L1 and L2 caches are modeled as SRAMs. The L3 cache is modeled as an eDRAM. Each L3 bank has a Mosaic module as in Fig. 3.5. A refresh operation is like a cache access, consuming an energy equal to a cache line hit. It takes one cycle when done in a pipelined fashion.

To evaluate Mosaic, we use a variety of tools. To estimate performance, we use the SESC [19] architecture simulator. To estimate the area and the dynamic and leakage energies of cores, caches and interconnect, we use McPAT [20]. Since McPAT does not model eDRAM caches, we use CACTI [18] to estimate the area and energy of the L3. We use the Synopsys Design Compiler to estimate the area and energy of the counters introduced by Mosaic. Finally, we use VARIUS [38] to model V_t variation and R [44] for fast statistical prototyping and analysis.

We target a platform where energy efficiency is critical. Hence, we use a modest

Table 3.3: Evaluation parameters & tools.

Architectural Parameters	
Chip	16-core CMP
Core	2-issue out-of-order
Instr. L1 cache	32 KB, 2 way. Round trip (RT): 2 ns
Data L1 cache	32 KB, 4 way, WT, private. RT: 2 ns
L2 cache	256 KB, 8 way, WB, private. RT: 6 ns
L3 cache	16 MB, 16 banks, WB, shared
L3 bank	1 MB, 8 way. RT: 14 ns (local), 26 ns (farthest)
Line size	64 bytes
DRAM	Idle RT from controller: $\approx$ 40 ns
Network	4 x 4 torus
Coherence	MESI directory protocol at L3
Step size	50 μs
Technology Parameters	
Tech. node	32 nm
Frequency	1000 MHz
Device type	Low operating power (LOP)
Temperature	330 Kelvin
eDRAM variation	$\mu(V_t)$=0.65V, $\sigma(V_t)$=0.042V (equal contrib. systematic and random), ϕ=0.4
Tools	
Architecture	SESC [19]
Time & Power	McPAT [20] and CACTI [18]
Synthesis	Synopsys Design Compiler
Statistics	R [44]
Variation	VARIUS [38]

frequency. We use area, energy, and timing estimates for 32 nm technology — although the T_{ret} distribution from IBM used in this proposal (Fig. 3.2) is for 65 nm technology. We use this distribution for lack of corresponding data at 32 nm. However, at 32 nm, the distribution changes only marginally [33]. To generate spatial maps for T_{ret}, we use the distribution parameters of Table 3.2. For each experiment, we average out the results of 20 T_{ret} maps. The V_t variation parameters used are shown in Table 3.3. The $\mu(V_t)$ and $\sigma(V_t)$ values are obtained from Kong et al. [8]. We assume an equal contribution of systematic and random components in $\sigma(V_t)$, but later we vary the breakdown.

We evaluate Mosaic designs with different combinations of tile size (T_{size}) and counter size. The parameter sweep is summarized in Table 3.4. A 1-bit counter

Table 3.4: Parameter sweep. We use 56 total combinations.

Tile size (lines)	1, 2, 4, 8, 16, 32, 64
Counter size (bits)	1, 2, 3, 4, 5, 6, 7, 8

rolls down every Step, and corresponds to the baseline (i.e., conventional) implementation of periodic refresh every Step. We assume a constant *T* of 330 K. We do not experiment with *T* variation (spatial or temporal) and the corresponding refresh rate adaptation.

We compare Mosaic against: (i) the baseline (i.e., conventional) periodic refresh, (ii) a proposed scheme that uses multiple refresh periods (RAIDR [40]), and (iii) an ideal design with the lower-bound refresh as given by Eq. 3.7. The ideal design is subjected to the guardband of Section 3.3.4, but not to the rounding-off constraint. Guardbanding is required because of port constraints, but rounding-off is an artifact of using counters.

For simplicity, we use a Step size of 50 μs. The baseline design refreshes at every Step and has no counter overhead. RAIDR [40] was proposed to refresh pages in DRAM main memories with bins that are in a geometric progression with a ratio of 2 i.e., either $64\ ms$, $128\ ms$ or $256\ ms$, depending on the pages' retention times. It uses a Bloom filter per bin. Since we apply it to eDRAMs, we set it to refresh cache lines at the Step size, 2 $\times$ the Step size, or 4 $\times$ the Step size — namely, $50\ \mu s$, $100\ \mu s$ or $200\ \mu s$. As we discuss in Section 3.7, we do not enable more bins to be faithful to the original design. In practice, more bins could be supported but, with many bins, the algorithm becomes inefficient: the bins for the higher refresh times quickly become too coarse-grained to be useful. Moreover, with many Bloom filters, the algorithm quickly becomes slow. For Mosaic and for the ideal design, we use fixed-distance bins: 50 μs, 100 μs, 150 μs, 200 μs, etc.

We evaluate these designs with the following 16-threaded applications from SPLASH-2 [23] and PARSEC, where the problem sizes are in parenthesis: FFT (2^{20}), LU (512x512), Radix (2M keys), Cholesky (tk29.O), Barnes (16K particles), FMM (16K), Radiosity (batch), Raytrace (teapot), Streamcluster (sim small), Blackscholes (sim medium), and Fluidanimate (sim small).

3.6 Evaluation

In Section 3.6.1, we evaluate the merit of several combinations of tile sizes and counter sizes in saving refresh energy. We choose the combination that minimizes the area and energy overheads of the counters, and maximizes refresh energy savings. In Section 3.6.2, we compare the resulting best Mosaic against the baseline, RAIDR, and ideal designs. We examine reduction in refreshes, system performance, and L3 energy savings. Finally, in Section 3.6.3, we perform a sensitivity analysis of the breakdown of $\sigma(V_t)$ into systematic and random components.

3.6.1 Finding the Best Mosaic

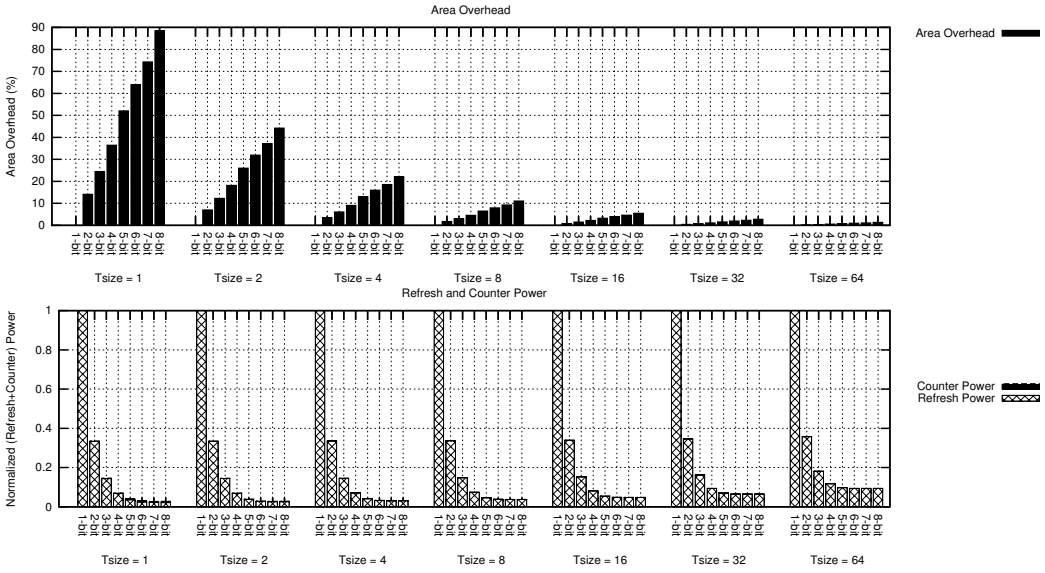


Figure 3.6: Area overhead of the counters (top), and power consumed refreshing L3 and operating the counters (bottom).

Figure 3.6 shows the area overhead of the counters (top), and the power consumed refreshing L3 and operating the counters (bottom) for different parameter combinations. In both plots, the X-axis is divided into 7 sets. Each set corresponds to a tile size (T_{size}) in number of L3 cache lines. Within each set, there are 8 bars for different counter sizes in bits. The area overhead of the counters is shown as a percentage of the L3 data array area. The power consumed is normalized to that of the baseline (i.e., conventional) design. It is broken down into contributions from refreshing the L3 cache, and operating the counters.

In both plots, across all tile sizes, a 1-bit counter is equivalent to not having a counter at all, and corresponds to the baseline. Therefore, in the area overhead plot, the 1-bit counter is marked zero. Likewise, in the power plot, its counter power component is zero. All the 1-bit combinations are equivalent and correspond to the baseline.

For a fixed T_{size}, the area overhead of the counters increases as the size of the counters increases. For a given counter size, its area overhead decreases as the T_{size} increases. This is because the same counter is now being shared amongst more lines.

For a given T_{size}, the refresh power decreases with the counter size. This is because the retention time of the tiles can be tracked at a much finer granularity. However, the benefits flatten out as the range of the counter approaches the maximum retention time of the distribution. As the T_{size} increases, the refresh power goes up. This is because all the lines in a tile are refreshed at the rate of the weakest line in the tile. We also see that the counter power is negligible compared to the L3 refresh power.

Therefore, there is a clear tradeoff in Mosaic between area overhead and refresh power savings. To help choose the best design, we only consider combinations with an area overhead of less than 2% and refresh power savings of at least 90%. Among the few candidate solutions, a tile size of 32 lines with a 6-bit counter is the best combination. It has an area overhead of 2% and refresh power savings of 93.5%. Henceforth, we call this combination the *Mosaic*.

3.6.2 Refresh Count, Performance & Energy

Figure 3.7 shows the total number of L3 refreshes (top), the execution time (center), and the L3 energy consumption (bottom) for different designs running the applications. In all plots, the X-axis is divided into 12 sets (11 for the applications and 1 for the average). Each application is run on the baseline, RAIDR, Mosaic, and ideal designs, and the result is normalized to the application's baseline design. In the L3 energy plot, each bar is broken down into dynamic, leakage and refresh energies from bottom to top. The dynamic energy is too small to be seen.

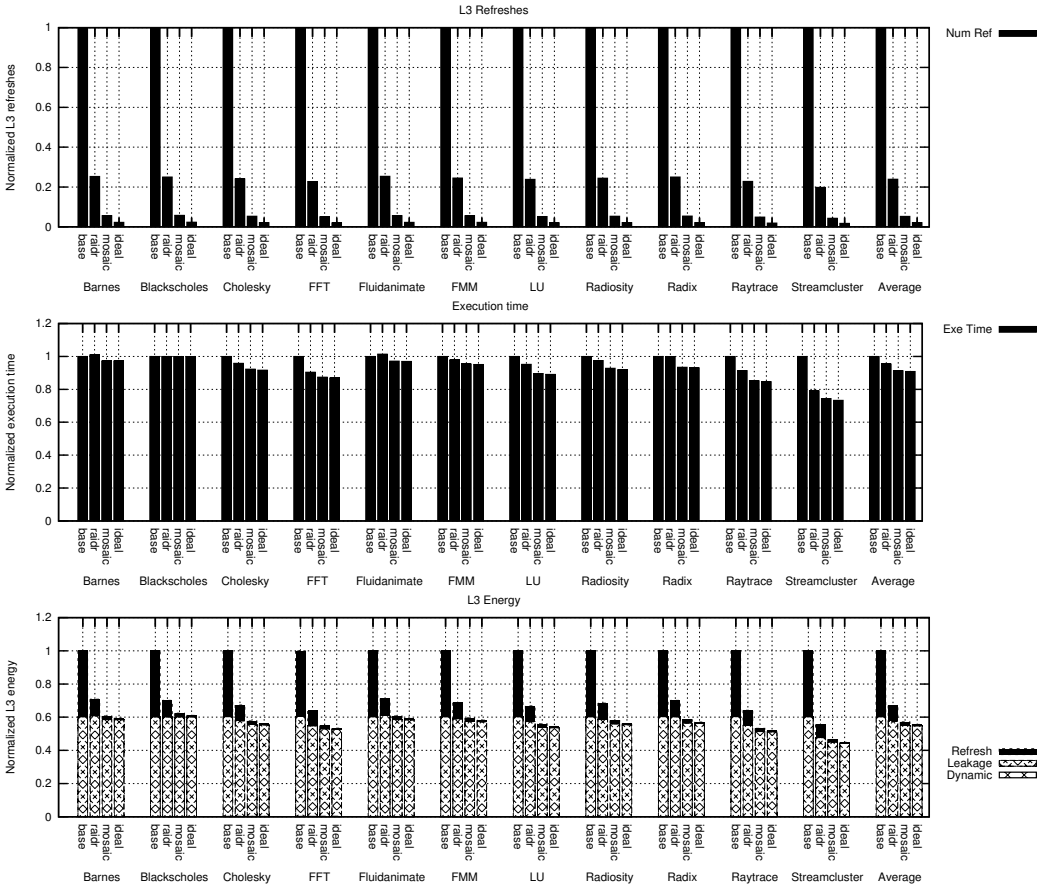


Figure 3.7: Number of L3 refreshes (top), execution time (center), and L3 energy consumption (bottom).

3.6.2.1 Total Refresh Count

As we go from baseline to RAIDR, to Mosaic, and to ideal, we see that the number of L3 refreshes decreases. There is little difference across applications; the difference is affected by how much the optimized designs speed up the particular application over baseline. Overall, on average, RAIDR reduces the number of L3 refreshes to a quarter (i.e., a reduction of $4x$). This is expected from the statistical distribution of T_{ret}, where most of the lines have a T_{ret} of over 200 μs. In contrast, on average, Mosaic reduces the number of refreshes by $20x$. In addition, Mosaic is within $2.5x$ of ideal. Recall that ideal has not been subjected to the rounding-off constraint. Any practical implementation, using counters or otherwise, will have additional overheads (e.g., area or precision loss), and will close the gap between Mosaic and ideal.

3.6.2.2 Performance

Across most applications, we see that the different optimized designs perform better than the baseline. The reason for the faster execution is that the reduction in the number of refreshes reduces L3 cache blocking. The applications with most L3 accesses are the ones that benefit the most. On average, RAIDR reduces the execution time by 5%, Mosaic by 9%, and ideal by 10%. Mosaic comes to within one percent of the execution time of ideal.

3.6.2.3 L3 Energy

Across all the applications, we see that the different optimized designs significantly reduce the L3 energy compared to baseline. The reduction comes from savings in refresh energy and (to a much lesser extent) leakage energy. As is generally the case for last level caches, the fraction of dynamic energy is very small. The savings due to refresh energy reduction are the most significant. The designs reduce refresh energy by significantly reducing the number of refreshes. We can see that Mosaic eliminates practically all of the refresh energy; its effectiveness is practically the same as the ideal design.

The leakage energy is directly proportional to the execution time. Since these optimized designs reduce the execution time, they also save leakage energy. Overall, on average, RAIDR saves 33% of the L3 energy. Mosaic saves 43% of the L3 energy and is within one percent of the ideal design.

3.6.3 Sensitivity Analysis

Up until now, we have assumed that $\sigma(V_t)$ has equal systematic and random components — i.e., $\sigma_{rand} : \sigma_{sys}$ is 1:1. In future technology nodes, the breakdown into systematic and random components may be different. Hence, we perform a sensitivity analysis, keeping the total $\sigma(V_t)$ constant, and varying its breakdown into the σ_{rand} and σ_{sys} components. We measure the power consumed by the Mosaic configuration chosen in Section 3.6.1, as it refreshes L3 and operates the counters. Fig. 3.8 compares the resulting power. The X-axis shows different designs, as we vary the ratio $\sigma_{rand} : \sigma_{sys}$, with the random component increasing to the right. The bars are normalized to the case for $\sigma_{rand} : \sigma_{sys} = 1 : 1$, and broken down into power consumed refreshing L3 and operating the counters.

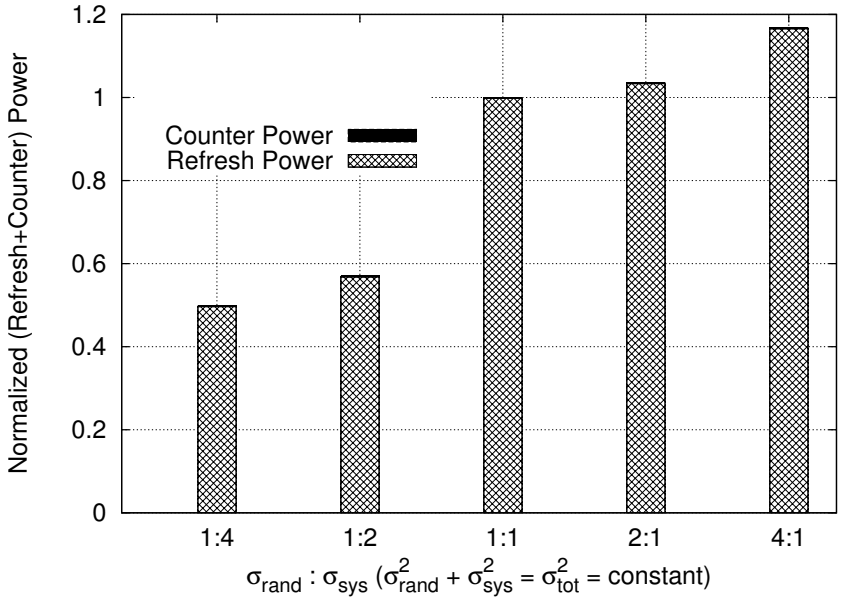


Figure 3.8: Power consumed refreshing L3 and operating the counters, as we change the breakdown of $\sigma(V_t)$.

The refresh power increases as the random component gains more weight. The reason is that, with relatively lower systematic component, the spatial correlation of T_{ret} decreases, eroding away some of the benefits of tiling. However, it is important to note that the increase in refresh needs is modest. Specifically, for a $\sigma_{rand} : \sigma_{sys} = 4 : 1$, the increase in power over the 1:1 configuration is only about 20%. With this increase, the power consumed refreshing L3 and operating the counters is still 92% lower than the baseline.

3.7 Related Work

Several approaches have been proposed to reduce the leakage (in SRAMs) or refresh (in eDRAMs/DRAMs) power in memory subsystems. One approach is to exploit the access patterns to the cache or memory. A second one is to take advantage of the intrinsic variation in the retention time of eDRAM lines or DRAM rows to save refresh power. A third one involves the use of error-correction codes (ECC) and tolerating errors.

As examples of the first class of approaches targeting SRAMs, we have Gated-V_{dd} [13] and Cache Decay [12, 16]. These schemes turn off cache lines that are not likely to be accessed in the near future, and thereby save leakage power.

Cache Decay relies on fine-grained logic counters, which are expensive, especially for large lower-level caches. Drowsy Caches [11, 17] periodically move inactive lines to a low power mode in which they cannot be read or written. However, this scheme is less applicable in deep-nm technology nodes, where the difference between V_{dd} and V_t will be smaller.

Ghosh et al. [30] propose Smart Refresh, which reduces refresh power in DRAMs by adding timeout counters per line. This avoids unnecessary refreshes of lines that were recently accessed. Refrint [45] uses count bits instead of counters to reduce the refresh power in eDRAM-based caches in two ways. First, it avoids refreshing recently-accessed lines. Second, it reduces unnecessary refreshes of idle data in the cache. Such data is detected and then written back to main memory and invalidated from the cache. Chang et al. [46] identify dead lines in the last-level cache (LLC) using a predictor and eliminate refreshes of it.

As part of the second class of approaches, there is work focused on reducing the refresh power of dynamic memories by exploiting variation in retention time. It includes RAPID [29], the 3T1D-based cache [26], and RAIDR [40]. RAPID [29] proposes a software-based mechanism that allocates blocks with longer retention time before allocating the ones with a shorter retention time. With RAPID, the refresh period of the whole cache is determined only by the used portion.

The 3T1D-based cache [26] is an L1 cache proposal that uses a special type of dynamic memory cell where device variations manifest as variations in the data retention time. To track retention times, the authors use a 3-bit counter per line, which introduces a 10% area overhead. Using this counter, they propose refresh and line replacement schemes to reduce refreshes.

RAIDR [40] is a technique to reduce the refresh power in DRAM main memories. The idea is to profile the retention time of DRAM rows and classify the rows into bins. A Bloom filter is used to group the rows with similar retention times. There are several differences between Mosaic and RAIDR. First, Mosaic observes and exploits the spatial correlation of retention times, while RAIDR does not. In DRAMs, an access or a refresh operates on a row that is spread over multiple chips, which have unknown correlation. Mosaic can be applied to DRAMs if the interface is augmented to support per-chip refresh.

Second, RAIDR classifies rows in a coarse manner, working with bins that are powers of 2 of the baseline (i.e., bins of *t*, *t*x2, *t*x4, *t*x8, etc.). Therefore, many bins are not helpful because the bins for the higher retention times quickly become too coarse-grained to be useful. Mosaic tracks the retention time of lines in a fine-

grained manner, using fixed-distance bins (i.e., t, tx2, tx3, tx4, etc.). This allows it to have tens of bins (64 with a 6-bit counter) and hence enables more savings in refresh power.

Finally, the RAIDR algorithm takes longer to execute with increasing numbers of bins. With 8 bins, in the worst case, it requires 7 Bloom filter checks for every line. Hence, RAIDR only uses 3 bins. The Mosaic implementation using a counter is simple and scalable.

The third class of approaches involves using ECC to enable a reduction in the refresh power [31]. ECC can tolerate some failures and, hence, allow an increase in the refresh time — despite weak cells. As a result, it reduces refresh power. One example of this approach is Hi-ECC [3], which reduces the refresh power of last-level eDRAM caches by 93%. The area overheads and refresh power reduction achieved by Hi-ECC and Mosaic are similar. However, Mosaic improves execution time by 9%, while Hi-ECC does not affect the execution time.

3.8 Summary

This proposal presented a new model of the retention times in large on-chip eDRAM modules. This model, called Mosaic, showed that the retention times of cells in large eDRAM modules exhibit spatial correlation. Based on the model, we proposed the simple Mosaic tiled organization of eDRAM modules, which exploits this correlation to save much of the refresh energy at a low cost.

We evaluated Mosaic on a 16-core multicore running 16-threaded applications. We found that Mosaic is both inexpensive and very effective. An eDRAM L3 cache augmented with Mosaic tiles increased its area by 2% and reduced the number of refreshes by 20 times. This reduction is 5 times that obtained by taking the RAIDR scheme for main memory DRAM and applying it to cache eDRAM. With Mosaic, we saved 43% of the total energy in the L3 cache, and got very close to the lower bound in refresh energy.

CHAPTER 4

XYLEM

Technology advances are about to enable further integration of computer architectures into 3D processor-memory stacks [4, 5, 6, 7]. 3D implementations offer several benefits, such as a reduction in interconnect length and power, smaller form factors, and support for heterogeneous integration. 3D integration is made possible by Die-to-Die (D2D) connections and Through Silicon Vias (TSVs). D2D connections, also called microbumps (or μbumps), exist between dies, while TSVs run across the thickness of a die.

In architectures that stack one processor die and multiple DRAM dies, thermal considerations are paramount [47, 48]. In these architectures, on-chip temperatures are higher than in conventional planar designs because of the increased transistor density and the higher inter-layer thermal resistance. In particular, DRAM dies will be especially affected for two reasons: (1) they will endure high temperatures because they are stacked with the processor die and (2) they will experience spatial temperature variations, the result of hot spots in the processor die and varying distance from the heat sink. This will make for a very different environment than in planar DRAMs.

It is well known that high temperatures are taxing for DRAM. At high temperatures, DRAMs will require circuit redesign and more expensive packaging solutions. Moreover, data retention decreases and, therefore, cells need more frequent refresh. Prior work has mitigated the effect of DRAM/eDRAM refresh in planar systems [49, 40, 30, 45, 3]. With DRAM integration into 3D processor-memory stacks, more aggressive refresh schemes are needed, to avoid high energy consumption and throughput loss.

There is much work to be done in the area of thermal management in 3D processor-memory stacks. Indeed, consider existing 3D DRAM standards, such as Wide I/O [50]. Its focus is on providing higher bandwidth and performance, not effective thermal management. It simply uses conventional thermal management techniques from planar memories like DDR [51, 52] and LPDDR [53], such

as adjusting the refresh rate depending on temperature. It does not have special mechanisms to deal with higher temperatures and spatial temperature variations.

While TSVs are used to carry electrical signals across dies, they are built with materials of high thermal conductivity. Therefore, researchers have proposed *Thermal* TSVs (TTSVs) to reduce temperatures in the stack [54]. TTSVs are dummy TSVs that are meant for thermal conduction, as opposed to electrical conduction. They take away heat from the active parts of dies and help reduce temperatures. They have great potential to help alleviate the thermal problem in 3D stacks.

Most prior work on thermal analysis of TTSVs has been done in the CAD area [55, 54, 56, 57, 58]. Such studies have usually focused on generic 3D stacks, typically ignoring the specific implementation constraints that processor and memory architectures impose on TTSVs. In addition, we find that they do not consider the high thermal resistance offered by the D2D layers. Such resistance can nullify any effect of TTSVs.

The goal of this proposal is to devise an effective scheme for TTSVs to reduce the steady-state temperature of the memory dies in 3D processor-memory architectures — by moving heat from the stack to the heat sink. Given the resulting temperature distributions, it is clear that existing refresh schemes for planar DRAM technologies are inadequate in this environment. Therefore, we also propose new DRAM refresh schemes for these architectures. The schemes take advantage of the spatial variation in temperature to reduce the number of refreshes.

Our proposal, called *Xylem*, has three aspects. First, we place the TTSVs subject to architectural constraints. Specifically, TTSVs have to respect the DRAM array structure and not go through a DRAM array. Moreover, DRAM manufacturers cannot anticipate the hotspots of the processor die that will be stacked with their DRAM dies and, therefore, have to provide a generic TTSV layout. Only under certain conditions are we able to co-design the processor and DRAM dies, and provide a customized TTSV placement.

The second aspect involves eliminating the paths of high thermal resistance that exist in the D2D layers of the stack. To this end, we propose to align and short the TTSVs in the back face of a die with μbumps. This creates a path of high thermal conductance without which TTSVs are useless.

The resulting temperature distribution still has a significant spatial variation, and therefore the third component is new DRAM refresh schemes for processor-memory stacks which can take advantage of this spatial variation in temperature.

We evaluate Xylem TTSV organizations and refresh schemes using simulations of a 3D processor-memory stack. Our baseline system is a Wide I/O-compliant stack that contains an 8-core processor die running at 3.1 GHz and 8 DRAM dies on top. We find that the best Xylem TTSV organization reduces the peak temperature of the DRAM stack by an average of 8.7 °C. Combined with the best DRAM refresh scheme we reduce the number of refreshes by an average of 85%.

Overall, the main contributions of this proposal are:

- The observation that, by carefully placing TTSVs, we can reduce the temperature of the memory dies.
- The insight that TTSVs are only effective if they are aligned and thermally shorted with μbumps.
- Generic and custom TTSV layouts for effective reduction of memory die temperatures, while abiding by architecture and implementation constraints.
- New DRAM refresh schemes for 3D processor-memory stacks that take advantage of the spatial variation in temperature to reduce refreshes.

This chapter is organized as follows: Section 4.1 provides a background; Section 4.2 has a tradeoff analysis of 3D organizations; Section 4.3 presents our Xylem proposal; Sections 4.4 and 4.5 model and evaluate it; and Section 4.6 describes related work.

4.1 Background

This section briefly describes 3D integration, TSVs, TTSVs, μbumps, 3D stacked memory standards, and existing refresh schemes.

4.1.1 3D Integration

3D integration of dies in a stack [4, 7] offers several benefits, including a reduction in interconnect length and power, smaller form factors, and support for heterogeneous integration [5, 6]. 3D integration can use die-to-die, die-to-wafer or wafer-to-wafer stacking.

The dies within a stack can be arranged in different configurations, such as: (i) active layer of the dies facing each other (face-to-face or f2f), (ii) active layer of one facing the bulk of another (face-to-back or f2b), and (iii) bulk layer of the dies facing each other (back-to-back or b2b). For multiple (more than 2) homogeneous dies, such as a stack of DRAM dies, f2b is an obvious choice. For multiple heterogeneous dies, such as memory and logic, charge-coupled device and logic, or analog and digital, the configuration is dictated by functionality, cost, performance, or thermal issues.

4.1.2 Through Silicon Vias (TSVs)

TSVs are vertical interconnects that run across the thickness of the die. There are several process flows for TSV fabrication, such as via first, via middle, frontside via-last, and backside via-last [59, 60, 61]. In addition, TSVs can be made before or after bonding. With the exception of the frontside via-last process, a TSV is present only in the silicon, and not in the metal layers of a die. The process flow chosen affects many aspects, including the metal routing, the choice of TSV metal (copper (Cu) or tungsten (W)), and the TSV dimensions.

Ideally, we want a high TSV density (i.e., the number of TSVs in a given area). The density is determined by the TSV's *aspect ratio* and the die thickness. A TSV's aspect ratio is its height divided by its diameter, and is determined by manufacturing constraints and choice of TSV metal. For example, W allows TSV aspect ratios of about 30:1, while Cu is limited to no more than 10:1. For a given die thickness, the TSV density is proportional to the square of the aspect ratio; hence, high aspect ratios are preferred. For a given aspect ratio, the density is inversely proportional to the square of the die thickness. Hence, to attain high densities (and to aid TSV fabrication), dies are thinned down to a few tens of microns [47, 48]. However, researchers have observed that die thinning reduces lateral thermal spreading and worsens chip temperatures [62, 58].

The choice of TSV metal is not trivial. Cu is a better electrical and thermal conductor than W. In addition, Si is about 30% harder than Cu but $5x$ softer than W. Therefore, under thermal stress, Si can prevent Cu from expanding, while W can crack the Si. However, as we noted, W allows higher TSV aspect ratios than Cu. The choice is also influenced by the temperatures encountered during chip fabrication. Emma et al. [62] compare Cu and W TSVs in detail.

4.1.3 Die-to-Die Vias or Micro-Bumps (μBumps)

μbumps provide the interconnection between the dies in the stack. They are like the C4 pads that connect a die to a board, but with much finer pitch. Their pitch is larger than that of TSVs, and hence they determine the interconnect density between the dies. The most widely-used implementation for a μbump is a Cu pillar with a tin-silver solder.

μbumps can be electrical or dummy. Electrical μbumps provide signal connections between the dies. To facilitate more electrical connections, electrical μbumps have a fine diameter and pitch of about 17 μm and 50 μm, respectively [63]. Dummy μbumps can be used for thermal conduction, ease of stacking, or mechanical support. They are not electrically or thermally connected to the adjoining layers.

4.1.4 Thermal Effects in 3D

3D stacking worsens on-die temperatures [47, 48, 58] because of the increased transistor density and higher inter-layer thermal resistance. The rate Q of heat flow (i.e., power) across a layer is proportional to the thermal conductivity of the layer (λ) and the temperature difference across the layer (δT). Mathematically, $Q \propto \lambda \times \delta T$. If the power dissipated in a layer is constant, then it follows that a lower λ results in a higher temperature difference across the layer. If the stack has multiple layers, then the temperature differences add up, and the layer farthest from the heat sink will be at high temperature.

4.1.4.1 Thermal TSVs (TTSVs)

Since TSVs are vertical metal interconnects with high thermal conductivity, researchers have proposed to use dummy TSVs simply for thermal conduction to reduce temperatures, as opposed to for electrical conduction. These are called Thermal TSVs (TTSVs). Placing TTSVs has been shown to improve heat transfer [55, 54]. Note that normal TSVs also aid in thermal conduction.

4.1.4.2 Dummy μBumps

In a 3D stack, the die-to-die (D2D) layer has the lowest thermal conductivity. This is because typical underfills have a $\lambda \approx 0.5$ W/(m·K). In contrast, silicon has a $\lambda \approx 120$ W/(m·K). Consequently, there is ongoing research on improving D2D layer conductivity, e.g., by improving underfill conductivity. An alternative is to fill the D2D layer with dummy μbumps, after provisioning for the electrical μbumps. Placing many μbumps has no area or manufacturing overhead.

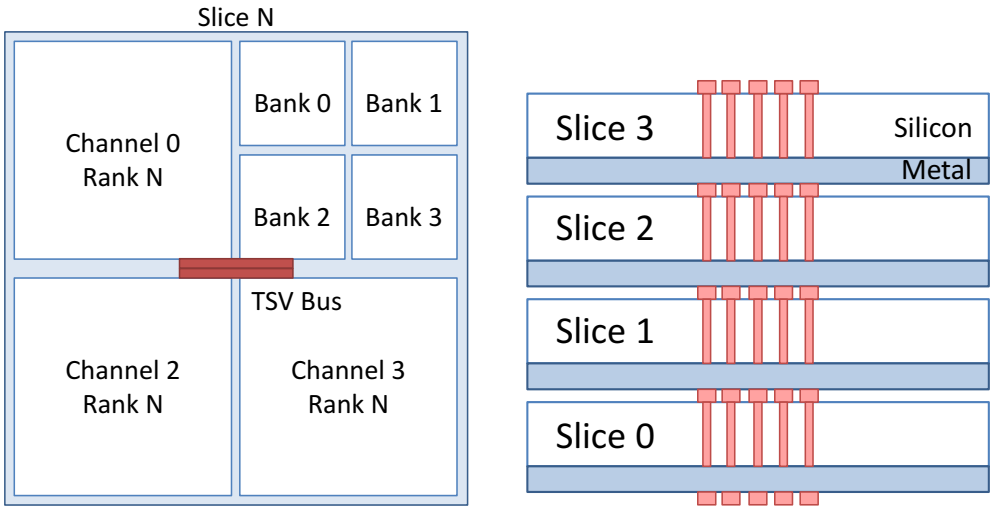
While the λ of just the Cu pillar with the solder is ≈ 40 W/(m·K), the real thermal conductivity of a D2D layer filled with dummy μbumps with a 25% density is *only* 1.5 W/(m·K) [64]. The reason is because the interconnection in a D2D layer includes multiple materials, including the solder with the Cu pillar, the Al pad, Cu, SiO_2 and SiN. A detailed cross section of the D2D interconnection layer can be found in [64]. Overall, since dummy μbumps increase the conductivity of the D2D layer, we assume them to be present in the rest of the proposal.

4.1.5 3D Stacked Memory Standards

Manufacturers and standardizing committees have proposed several 3D-stacked memory architectures. They include Hybrid Memory Cube (HMC) [65] from Micron, and High Bandwidth Memory (HBM) [66] and Wide I/O [50] from JEDEC [67]. HMC and HBM are stacks of memory dies with a logic chip (optional) for controller operations. Wide I/O (prototyped by Samsung [63]) is a memory-only stack which can be connected to the processor die through TSVs.

Fig. 4.1 shows the Wide I/O organization. It supports 4 physical channels. Each physical channel contains independent control, data, and clock signals. Each memory die in the stack is called a slice, and has 4 ranks (1 per channel). A stack of 4 slices results in 4 ranks per channel and 16 overall. Each rank is further divided into 4 banks, resulting in a total of 16 banks per slice (4 per channel). Altogether, for a stack of 4 slices, each channel has 16 banks and the entire stack has 64 banks.

The Wide I/O standard supports very modest signaling rates of 200 MHz and 266 MHz at Single Data Rate (SDR). The peak bandwidth of all channels combined is 12.8 GB/s. JEDEC has a roadmap [68, 69] for increasing the signaling rates and using Dual Data Rate (DDR) for Wide I/O 2. This will increase the bandwidth to about 128 GB/s.



(a) Slice organization.

(b) Stack organization.

Figure 4.1: Wide I/O organization (not drawn to scale).

4.1.6 Refresh Schemes in DDRx/LPDDRx

DDR2 [51] and DDR3 [52] perform per rank refresh i.e. a refresh command from the memory controller to the rank refreshes all banks in the rank (4 in DDR2 and 8 in DDR3). Before a rank refresh command can be issued all banks comprising the rank have to be in the precharged state.

LPDDR3 [53] is an extension of DDR3 with more power management features. In particular, w.r.t. refresh, it supports per bank refresh, i.e. it allows only one bank to be refreshed at a time. Before a bank refresh command can be issued the bank should be in the precharged state. This reduces blocking due to refreshes as only one bank is unavailable. The banks in a rank have to be refreshed in a round-robin fashion.

Both schemes (per rank, per bank) use the 'Auto Refresh' command. In 'Auto Refresh' the memory controller only initiates the refresh, while the row addresses are maintained internally in the banks. However, in both schemes the refresh rate is a constant, i.e. all banks in a channel have to be refreshed at the same rate.

4.1.6.1 Temperature Sensing & Adaptation

Memory dies detect temperatures with a high accuracy using on-die temperature sensors. Temperature sensors occupy very small area and have very low power overheads. Kim et al. [70] implemented a temperature sensor for DRAMs in

0.016 mm^2 with a resolution of 0.7 °C which consumes under 1 μW of power when used at a rate of 1 sample per second.

In a 3D processor-memory stack the location of hotspots cannot be predicted by the memory manufacturers. Therefore, in order to provide flexibility Wide I/O supports several locations for the placement of temperature sensors on die. The specific location to be used for thermal sensing can be set using a configuration register.

DRAM refresh cycle is exponentially dependent on temperature. Current memory dies have mechanisms to perform Temperature Compensated Refresh (TCR) and Temperature Compensated Self Refresh (TCSR), where the refresh cycle is halved for every 10 °C increase in temperature. Typically, for DDRx devices the refresh cycle is 64 ms at 85 °C and is halved or doubled depending on the direction of temperature change. If T is the temperature in Celsius, then the refresh cycle, T_{ref} in milliseconds is given by

$$T_{ref} = 23156.8 \times 10^{(-0.03010 \times T)} \tag{4.1}$$

The operating temperature and the refresh cycle values allowed by current standards is summarized in Table 4.1.

Table 4.1: Operating Temperatures and Refresh Cycle Values.

Standard	Temperature	Refresh Cycle
DDR3	$\leq$ 85 °C	64 ms
	85 - 95 °C	32 ms
DDR4	$\leq$ 85 °C	64 ms
	85 - 95 °C	32 ms
LPDDR2	$\leq$ 65 °C	256 ms
	65 - 75 °C	128 ms
	75 - 85 °C	64 ms
	85 - 105 °C	16 ms
LPDDR3	$\leq$ 65 °C	256 ms
	65 - 75 °C	128 ms
	75 - 85 °C	64 ms
	85 - 95 °C	32 ms
	95 - 105 °C	16 ms
Wide I/O	$\leq$ 85 °C	64 ms
	85 - 95 °C	32 ms
	95 - 105 °C	16 ms

4.2 Trade-offs in Stack Organizations

When integrating multiple DRAM dies and a processor die in a stack, there are conflicting manufacturability and thermal considerations, which lead to "processor-on-top" and "memory-on-top" organizations. These organizations are shown in Fig. 4.2a and 4.2b, respectively.

In either organization, the top die is connected to the heat sink and the Integrated Heat Spreader (IHS) using a Thermal Interface Material (TIM) [71]. The IHS is only present in desktops/servers, and not in mobile devices like laptops. Then, the different memory and processor dies are connected to each other through D2D connections and TSVs.

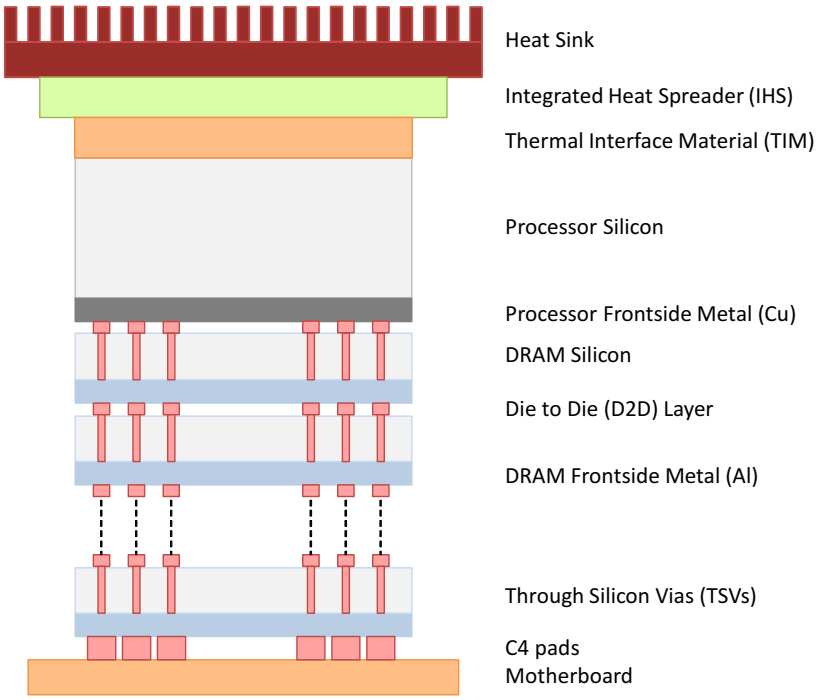
4.2.1 Processor on Top

The "processor-on-top" organization (Fig. 4.2a) has thermal advantages but significant manufacturing limitations. The advantages come from placing the die with most of the power dissipation closest to the heat sink. Also, the frontside metal layer of the processor die faces the memory stack, so that the processor die does not require TSVs or die thinning. The memory dies have TSVs and require die thinning.

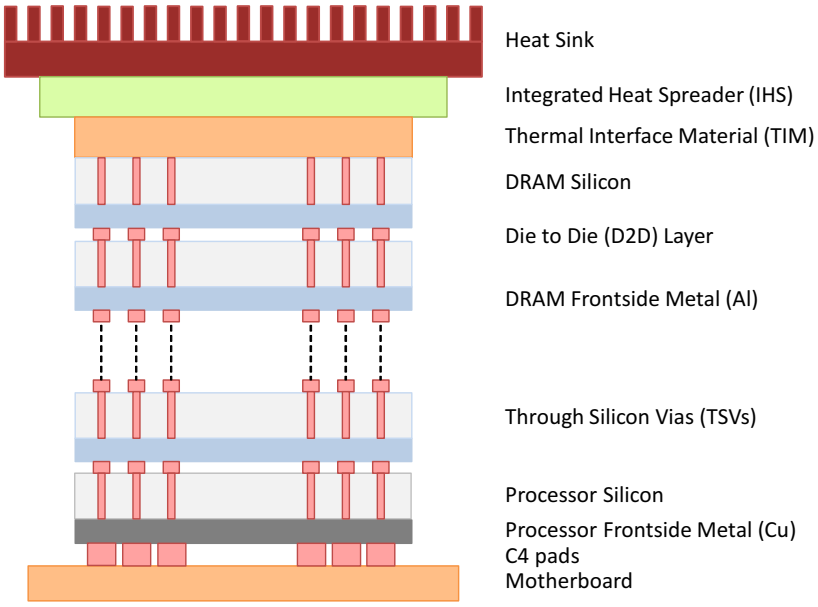
The manufacturing difficulties result from the fact that a typical processor die has close to a thousand pins, about half of which are devoted to power and ground signals [62, 72]. In this organization, the memory dies have to provision TSVs to connect the processor power, ground and I/O signals to the C4 pads. This is a large overhead. Moreover, different processors have different pin number and location requirements. Therefore, the memory vendor either has to grossly over provision TSVs to accommodate a wide variety of processor dies or manufacture custom dies for different processor vendors. Neither approach is desirable. In addition, TSVs add resistance to the Power Delivery Network (PDN) [62, 72]. For a current-hungry processor die away from the C4 pads, the IR drop across the TSV is a concern.

4.2.2 Memory on Top

The "memory-on-top" organization (Fig. 4.2b) has some thermal challenges, but clear manufacturing advantages. The thermal challenges result from the fact that



(a) Processor on Top Stack Organization.



(b) Memory on Top Stack Organization.

Figure 4.2: 3D stack organizations.

the heat generated in the processor die has to traverse the entire stack to reach the heat sink. Hence, the processor die will be at a significantly higher temperature than in planar organizations.

In this design, the frontside metal layer of the processor die is adjacent to the

C4 pads. The main manufacturing advantage is that the high current-carrying power and ground signals and the high-frequency I/O signals do not traverse any TSVs, and avoid the IR drop issue mentioned above. The memory stack in this configuration only contains TSVs as defined by the various 3D memory stacking standards, such as Wide I/O. Also, the processor die only has to provision for the number and location of signals required for stack integration as given by the memory-stacking standards. Overall, the memory and processor die floorplan are independent of each other.

In this design, the processor die includes TSVs and requires thinning. While the top-most memory die does not require TSVs, we add them since all memory dies are fabricated using the same steps. Overall, given the manufacturing advantages of the "memory-on-top" organization, it is the preferred one.

4.3 Placement, Alignment, & Shorting of TTSVs

In 3D processor-memory stacks DRAM dies will experience high temperatures because of their proximity to the processor die. They will also experience spatial temperature variations, the result of hotspots in the processor die and varying distance from the heat sink. As is known, high temperatures are taxing for DRAM. Refresh rates are exponentially dependent on the die temperature. Given this direct connection between temperature and refresh in 3D stacks, we propose strategically *placing* TTSVs, and then *aligning, and shorting* them with μbumps to reduce memory temperature. The resulting temperature distribution still has significant spatial variation. We propose smarter refresh schemes to exploit this distribution. The result is substantial reduction in the number of refreshes.

Our proposal has three parts. The first involves placing the TTSVs in the best place on die, subject to certain constraints (Sections 4.3.1 and 4.3.2). The second involves aligning and shorting the TTSVs with μbumps, to avoid paths of high thermal resistance in the D2D layers (Section 4.3.3). The third component is new DRAM refresh schemes for processor-memory stacks.

4.3.1 TTSV Placement Constraints

Fig. 4.1 showed a die and a stack that follow the Wide I/O organization. Each die has a TSV bus only in the center of the die. These are 1,200 TSVs meant for elec-

trical connections. Therefore, there is an opportunity to place TTSVs distributed throughout the die to reduce the temperature of the memory dies. However, the placement of TTSVs has to abide by floorplan and physical constraints. We discuss these constraints next.

4.3.1.1 Floorplan Constraints

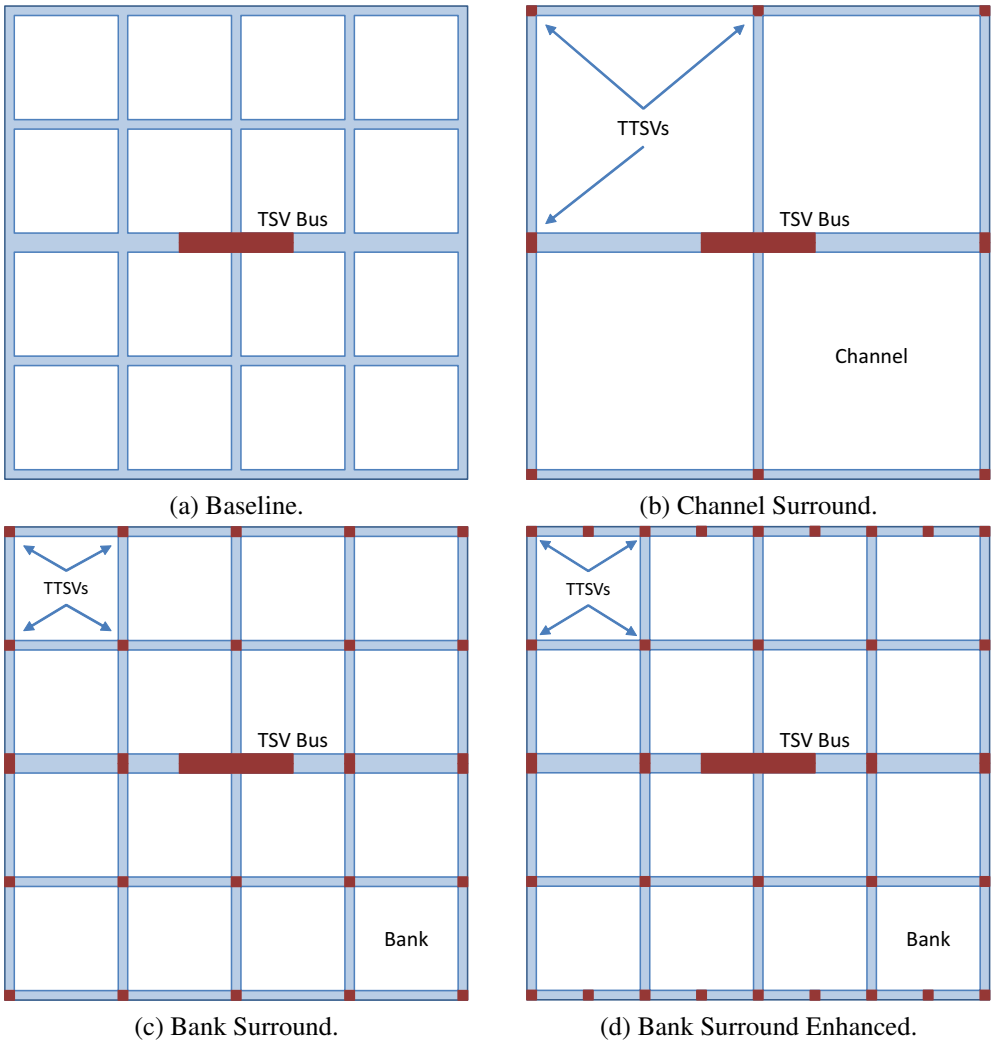
As can be deduced from the die floorplan of Fig. 4.1a, the majority of the DRAM die area is occupied by memory banks. Memory banks are very regular structures and are optimized for density and performance. Electrical or thermal TSVs should not disrupt the regular nature of these blocks. Therefore, we are limited to placing TTSVs in the area between the banks. This area is used for peripheral logic, such as row and column decoders, input/output buffers, testing circuitry etc. This peripheral logic is not very regular and, therefore, it is relatively easy to find dead spaces for TTSV insertion.

4.3.1.2 Physical Constraints

The placement of TTSVs is also affected by a few physical constraints that affect TSVs. One such constraint is due to the TSV's lateral thermal blockage. Specifically, as observed by Chen et al. [73], groups of TSVs (also called TSV *farms*) block horizontal (i.e., lateral) heat flow in thinned dies causing hotspots. This is in contrast to TSV's ability to aid vertical heat flow. Consequently, we should distribute the TTSVs, instead of aggregating them into a large TTSV farm.

A second constraint appears because TSV fabrication (both electrical and thermal) causes tensile stress around TSVs [74], which impacts transistor behavior in their neighborhood. A conservative way to reduce a TSV's impact on nearby logic is to create a *Keep Out Zone* (KOZ) around it.

Note that another reason to create a KOZ around a TSV is to minimize capacitive coupling and cross talk effects from neighboring TSVs, as well as from metal wires in planar dies [75]. However, TTSVs are electrically neutral and, therefore, not affected by this issue.



(a) Baseline. (b) Channel Surround.

(c) Bank Surround. (d) Bank Surround Enhanced.

Figure 4.3: TTSV placement schemes.

4.3.2 Proposed TTSV Placement Schemes

Based on the floorplan and physical constraints just described, we propose several TTSV placement schemes. We build on top of the baseline floorplan of Fig. 4.3a, which has a TSV bus of 1,200 TSVs in the center of the chip. Although these are TSVs for electrical conduction, they also aid in thermal conduction. Our proposals fall into generic and custom placement schemes.

Ideally, TTSV placement should be generic. Memory manufacturers should not make assumptions about other layers of the stack, such as the location of the processor die and its hotspots. We propose two generic TTSV placement schemes.

In *Channel Surround* (Fig. 4.3b), we place TTSVs in the peripheral logic at the vertices of each channel. Note that the peripheral logic area that runs horizontally

across the die center is wider than elsewhere because of the Wide I/O 12x100 TSV bus. Hence, we place two TTSVs in the center stripe instead of one. Overall, the total number of TTSVs in the die is 10. This design satisfies the floorplan constraint of not disrupting regular DRAM structures and the physical constraint of preventing lateral thermal blockage.

In *Bank Surround* (Fig. 4.3c), we place TTSVs in the peripheral logic at the vertices of each bank. Overall, the total number of TTSVs in the die is now 28. This design also satisfies the same constraints as Channel Surround, but distributes the TTSVs in a more fine-grained manner.

While these two schemes are effective, they do not exploit processor hotspot information. If we know the hotspots in the processor, we can devise a custom TTSV placement strategy. Such placement is likely to be more effective.

For our custom TTSV placement scheme, we use the processor die floorplan shown in Fig. 4.4. This is a typical layout for commercial processors [76, 77, 78, 79, 80, 81], where the cores are on the outside and the Last-Level Cache (LLC) in the center. This layout minimizes the LLC access time and the non-uniform access effects. In addition, it separates the hot spots, which are the cores.

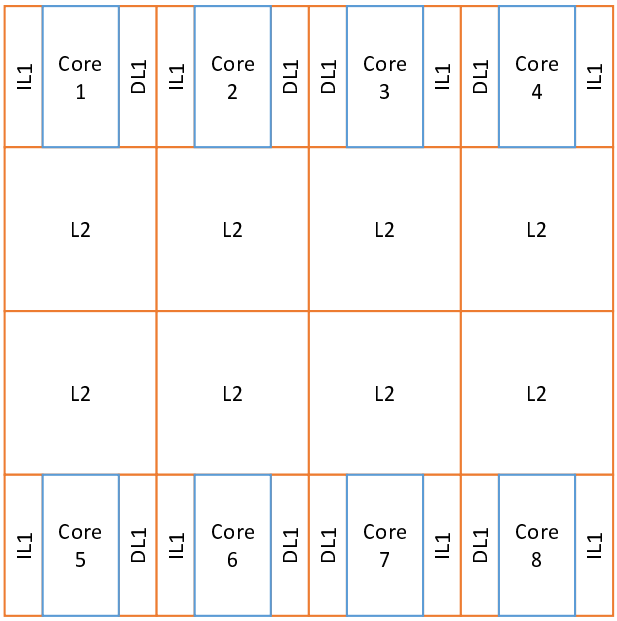


Figure 4.4: Processor die floorplan.

Knowing the location of the cores, we add eight additional TTSVs close to the cores. The result is the *Bank Surround Enhanced* scheme of Fig. 4.3d. The total

number of TTSVs is now 36. In effect, we have co-designed the memory and processor dies.

As we go from Channel Surround to Bank Surround and Bank Surround Enhanced, the schemes have more TTSVs. However, we will see in our evaluation that, even in the scheme with most TTSVs (Bank Surround Enhanced), the area taken by the TTSVs and KOZs is only 0.80% of the total die area. Moreover, we will also examine the effect of TTSV location for a constant TTSV count.

4.3.3 TTSV-μBump Alignment and Shorting

To attain effective heat transfer from the stack to the heat sink, it is not enough to provide TTSVs. In addition, there should not be any layer in the stack with low thermal conductivity. Unfortunately the D2D layers in Fig. 4.2b have low thermal conductivity. In this section, we describe why they do and our proposal to get around the problem. Surprisingly, this issue has been largely ignored in previous studies and is discussed in Sec 4.6.

4.3.3.1 Low-Conductivity D2D Layer

Figure 4.5 shows the interface between two f2b DRAM dies [82]. From top to bottom, we see the face side of a die, the D2D layer, and the back side of another die. The silicon layer of the top die contains the active devices and electrical TSVs. The frontside metal layers of the top die contain metal routing layers (M_1 to M_n) separated by low thermal conductivity dielectric materials. The D2D layer consists of the μbump layer and the backside metal layers of the bottom die. The latter typically have 0 to 2 layers of backside metal routing (BM_1 to BM_n). On the left side, we see a TSV in the bottom die connected through the backside metal layers to an electrical μbump and then through the frontside metal layers to a device in the top die.

In the figure, we have added two TTSVs and two dummy μbumps (shown in stripes). As we can see in the figure, the TTSVs typically terminate at the silicon layers. In particular, they avoid the frontside metal layers because they would cause routing congestion in the already busy metal layers. [1] Moreover, the TTSVs

[1] The exception is when TSVs are built using the more costly frontside via last process, as in Black et al. [47].

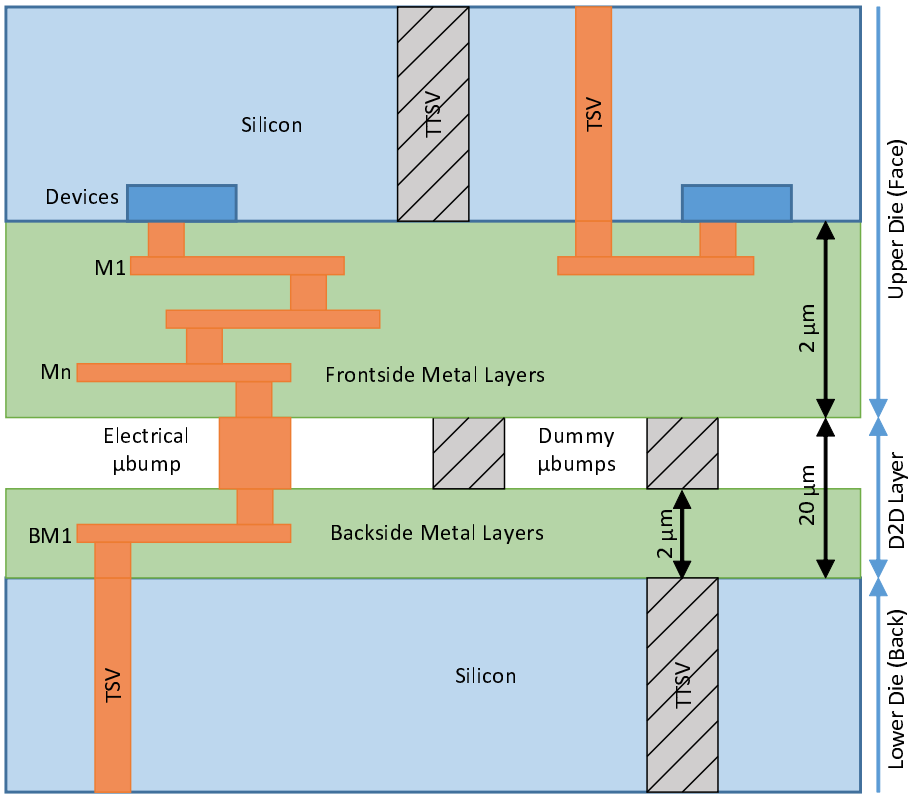


Figure 4.5: Interface between two f2b DRAM dies. The thickness of the layers is not drawn to scale.

in the top and bottom dies do not need to be vertically aligned with each other or the μbumps.

Prior work explicitly assumed that the D2D layer has a reasonably high conductivity or small thickness. For example, Loh [48] models the D2D layer as 25% Cu and 75% air, and uses a thermal conductivity of $\lambda \approx 100$ W/(m·K). Also, Goplen et al. [54] model the thickness of the D2D layer to be only 0.7 μm.

However, recent data shows that the D2D layer has low thermal conductivity. Matsumoto et al. [64] measured the conductivity of a D2D layer composed of a metal routing layer and Cu-Sn-Ag based μbumps. While the conductivity of just the μbump is $\approx$ 40 W/(m·K), the experimentally measured conductivity of the whole D2D layer is *only* $\approx$ 1.5 W/(m·K). Also, the thickness of the D2D layer is about 20 μm [64].

The thermal resistance per unit area (R_{th}) of a layer is the layer's thickness divided by its conductivity. The D2D layer has a high R_{th} because it has a sizable thickness ($\approx$ 20 μm) and a low thermal conductivity (λ = 1.5 W/(m·K)). Specifically, its R_{th} is $\approx$ 13.3x10^{-6} m^2·K/W. In comparison, the R_{th} of the silicon layer

is $\approx$ 0.83x10^{-6} m^2·K/W. The presence of such a layer between the TTSVs in two adjacent dies will in practice zero out any positive impact of the TTSVs. Our experiments validate this observation.

4.3.3.2 Proposed Solution

Our goal is to reduce the R_{th} of the D2D layer. To do so, we propose to (1) *align* TTSVs in the back side of a die with dummy μbumps in the D2D layer, and (2) short the TTSVs and the dummy μbumps.

To see how this is done, consider Figure 4.5. In the figure, the TTSV in the bottom die is aligned with a dummy μbump. Then, using backside metal vias we short the TTSV with the dummy μbump. Ideally, we would want to do the same operation with the TTSV in the top die. However, the frontside metal layers carry many electrical signals and using a via stack of frontside metals may or may not be possible and may cause routing congestion. Consequently, we neither align nor short the TTSVs from the face side. Note that for a stack of memory dies the TTSVs will be aligned because the die floorplans are the same.

To get an idea of the resulting D2D layer's R_{th}, assume that we did align and short TTSVs from both dies with μbumps. TTSVs are very good thermal conductors. Given that the thermal conductivity of the μbump is $\approx$ 40 W/(m·K) [64] and its thickness is $\approx$ 18 μm, the D2D layer's R_{th} is now $\approx$ 0.45x10^{-6} m^2·K/W in the neighborhood of the μbump. This is $30x$ lower than before. In practice, since we only align and short one of the two sides, we do not get all of this improvement. However, we reduce the R_{th} by close to one order of magnitude, which is very effective.

Since dummy μbumps are plentiful, this alignment is not difficult at all. Ideally, we want the TTSV diameter to be the same as the μbump diameter, to facilitate maximum heat flow.

Finally, electrical TSVs also contribute to thermal conduction because they are connected to μbumps. However, their contribution is limited. The reason is that the placement of electrical TSVs in the chip is dictated by stacking standards. For example, Wide I/O requires all 1,200 TSVs to be clustered together in the center of the memory die. As a result, they are unlikely to be all aligned to electrical μbumps. Instead, as shown in Figure 4.5, they are connected to μbumps using thin backside metal routing wires, which results in higher thermal resistance.

4.3.4 Refresh Schemes for 3D Processor-Memory Stacks

The placement of TTSVs, and their alignment and shorting with dummy μbumps increases the heat flow from the stack to the heat sink, which causes a temperature reduction in the memory dies. Existing refresh schemes for planar organizations like DDRx and LPDDRx, and 3D organizations like Wide I/O adapt refresh rates with temperature and will take advantage of this reduction. However, they are too conservative. They do not take into account the thermal environment of a processor-memory stack where different ranks and banks of a channel in the same stack could be at significantly different temperatures (unlike the planar environment).

As discussed in Section 4.1.6, existing standards refresh all ranks and banks in a channel at the same rate. They are refreshed at the rate dictated by the maximum temperature observed in the channel. This policy is clearly too conservative and increases the number of refreshes performed.

In addition, the existing adaptation is too coarse grained in the temperature domain. As listed in Table 4.1, the refresh rates are adjusted in steps of 10 $^{\circ}$C increase or decrease of temperature. For example, even if the temperature is 86 $^{\circ}$C the memory will be refreshed with a cycle time of 32 ms (for DDR3, DDR4, LPDDR3, Wide I/O) or 16 ms (for LPDDR2). This again is too conservative and increases the number of refreshes performed. This discretization affects both planar and stack memories.

To summarize, existing refresh schemes are too conservative for 3D processor-memory stacks both in the spatial domain and in the temperature domain. Therefore, we propose the following refresh schemes for processor-memory stacks.

Independent Per Bank Refresh To exploit the spatial variation (both horizontal and vertical) of temperature, we propose independent per bank refresh such that each bank can be refreshed at its own rate.

Continuous Temperature Adaptation To reduce the overheads introduced by temperature discretization, we propose continuous temperature adaptation.

4.3.4.1 Memory Controller Overheads & Operation

As mentioned in Section 4.1.6, LPDDR3 already supports per-bank refresh. However, the banks have to be refreshed at the same rate and in a round robin fashion.

We relax this constraint and allow each bank to be refreshed at its own rate. Moreover, this mode of refresh is still compatible with the 'Auto Refresh' command where the refresh operation is initiated by the memory controller, while the row addresses are maintained internally in the banks.

On-die temperature sensors in the memory die have a resolution of $\pm$ 1 $^{\circ}$C [83]. As mentioned in Section 4.1.6, temperature sensors occupy a small area and have very low power overheads. Using a temperature sensor per-bank has an area overhead of about 0.256 mm^2 (per die) and a power overhead of under 16 μW. These are small compared to the 64.34 mm^2 memory die area [63] and the power dissipated in a memory slice (several hundreds of mW).

To support independent per-bank refresh and continuous temperature adaptation, the memory controller has to maintain per-bank counters and per-bank temperatures. The memory controller also maintains a Look-Up-Table (LUT) which translates temperatures to refresh interval (t_{REFI}). A stack with 4 slices has 64 banks. The counter and temperature can be maintained in 2 registers, resulting in 128 registers overall. The counters are decremented every clock cycle. On reaching a value of zero, they interrupt the memory controller. The memory controller then schedules a refresh to that bank. It also probes the bank temperature, performs a table look up and loads the refresh interval value into the counter.

In addition, the memory controller has to use a guard band to account for the delay between temperature sensing and refresh rate adjustment. For example, LPDDR3 [53] uses a temperature margin of 2 $^{\circ}$C for temperature adaptation. The guard band can be small because temperature changes slowly while the entire bank is refreshed multiple times every second ($1/64\ msec \approx 16$). Since temperature changes slowly, the memory controller can perform temperature adaptation less often, say once every few seconds.

4.4 Modeling a 3D Processor-Memory Stack

4.4.1 Thermal Modeling

We model a "memory-on-top" 3D processor-memory stack as being composed of 8 distinct layers, namely the heat sink, Integrated Heat Spreader (IHS), Thermal Interface Material (TIM), DRAM silicon, DRAM metal, D2D layer, processor silicon, and the processor metal layer (Fig. 4.2b). Some of the layers occur multiple

times in the stack. TTSVs are present in the DRAM silicon and in the processor silicon.

4.4.1.1 Silicon & Frontside Metal

For accurate thermal analysis, the silicon and metal layers of a die are modeled as two separate layers [47, 48]. We model the metal layers of the die to include the active silicon in the same layer because it is difficult to separate the power into that consumed by transistors and that consumed by wires. The metal routing layers — typically Al for memory and Cu for processor — with dielectrics have a different thermal conductivity (λ) than silicon. Since Al has a lower λ than Cu, the λ of the metal layer of a memory die is lower than that of the metal layer of the processor die [47].

4.4.1.2 D2D Layer

The D2D layer contains μbumps and backside metal layers, and is modeled as discussed in Sec. 4.3.3. Electrical μbumps in the center of the die have a size and pitch of 17 μm and 50 μm, respectively [63]. The remaining area is filled with dummy μbumps of size 100 μm and 25% occupancy.

4.4.1.3 TSVs & TTSVs

We use Cu as the material for electrical TSVs and TTSVs because it has a high electrical and thermal conductivity. From ITRS [84], the electrical TSV size is 10 μm. Since the aspect ratio of Cu is limited to 10:1 [62], we use a die thickness of 100 μm for all the dies in the stack. We use a KOZ of 10 μm in both the X and Y dimensions. This results in an X and Y pitch of 20 μm, and fractional area occupancies of 0.25 and 0.75 for the Cu and Si, respectively. TTSVs have the same size as dummy μbumps and a KOZ of 10 μm in both X and Y dimensions.

The presence of TSVs (electrical or thermal) makes the corresponding layer heterogeneous — i.e., different blocks within the same layer have different λ. The λ of a block with two different materials (A and B) with conductivities λ_A and λ_B and fractional area occupancies ρ_A and ρ_B, such that $\rho_A + \rho_B = 1$, is [85]: $\lambda = \rho_A \times \lambda_A + \rho_B \times \lambda_B$.

For example, in the DRAM and processor silicon layer, there are blocks with just silicon, blocks with just TTSV and blocks with a TSV bus. A block with silicon has a λ = 120 W/(m·K). A TTSV block has a λ = 400 W/(m·K). A block with a TSV bus has a $\lambda = \rho \times 400 + (1\text{-}\rho) \times 120 = 190$ W/(m·K). The final λ of the layer depends on what fraction of the area is taken by the blocks of each kind. Similarly, the D2D layer has a λ = 1.5 W/(m·K), although the μbump itself offers a λ = 40 W/(m·K).

Table 4.2 shows the dimensions and λ of the various layers and blocks in the stack. These values are obtained from various sources [47, 62, 86, 48].

Table 4.2: Dimensions and thermal parameters.

Layer	Dimensions	Thermal Conductivity (W/(m·K))
Heat Sink	6.0x6.0x0.7 cm^3	400
IHS	3.0x3.0x0.1 cm^3	400
TIM	50 μm	5
DRAM Silicon	100 μm	120 (Si); 400 (TTSV); 190 (TSV bus)
DRAM Metal	2 μm	9
D2D	20 μm	1.5; 40 (μbump)
Proc Silicon	100 μm	120 (Si); 400 (TTSV); 190 (TSV bus)
Proc Metal	12 μm	12

Table 4.3 shows the different TTSV placement schemes evaluated. For each scheme, we list the name used in the evaluation and the number of TTSVs per chip. Most of the evaluation compares the top four schemes, which have a different TTSV count per chip. We also compare Channel Surround to a new scheme (*peri*) that we will describe. It has the same total TTSV count but a different placement of TTSVs.

Table 4.3: TTSV placement schemes evaluated.

TTSV Placement Scheme	Name	#TTSVs per Chip
Baseline	*base*	0
Channel Surround	*chan*	10
Bank Surround	*bank*	28
Bank Surround Enhanced	*banke*	36
Periphery	*peri*	10

4.4.2 Architecture

We evaluate Xylem on an 8-core chip multiprocessor (CMP) below a stack of 8 DRAM dies. Each core is a four-issue, out-of-order engine. It has a private instruction cache (IL1), a private data cache (DL1) and a private unified second-level cache (L2). A snoopy MESI protocol is employed to maintain coherence between the L2s. We use a maximum safe operating temperature for the memory (T_{max}) of 105 °C.

Our DRAM stack organization closely follows the JEDEC specification of Wide I/O [50]. Note that, although we use Wide I/O for evaluation, the problems and solution proposed are generally applicable to any processor-memory stack. As pointed out before, the existing Wide I/O standard supports very modest signaling rates of 200 MHz and 266 MHz at SDR. JEDEC has a road map for increasing the signaling rates and using DDR for Wide I/O 2 [68, 69]. Hence, in our experiments, we use the Wide I/O stack organization but use signaling rates of 800 MHz with DDR similar to DDR3-1600.

We organize the 3D stack of a processor die and 8 memory dies with almost the same area and aspect ratio. Our chip area and current DRAM densities allow each of the 8 DRAM dies to have 4 Gb (512 MB) of memory, resulting in 4 GB of memory for the stack. The system can cycle between 2.4 GHz (default) and 3.5 GHz in 100 MHz steps. The architectural parameters are summarized in Table 4.4.

4.4.3 Tools and Applications

We use a few tools to model the architecture, estimate area, power and timing, develop a floorplan, and model thermal affects. We use the SESC [19] cycle-level simulator to model the architecture. We obtain the dynamic and leakage energy of the processor die from McPAT [20]. The timing and energy of the memory dies is modeled with DRAMSim2 [87, 88]. We extended DRAMSim2 to support independent per bank refresh such that each bank can be refreshed at a different rate.

We use McPAT to estimate the area of the blocks within the processor die. The floorplan for each layer in the 3D stack is obtained using ArchFP [89, 90]. We ensure that known hotspots in the processor die such as FPUs are spatially separated from each other. Fig. 4.4 shows the processor-die floorplan.

Table 4.4: Architectural parameters.

Processor Parameters	
Multicore chip	Eight 4-issue out-of-order cores
Instruction L1	32 KB, 2 way, 2 cycles Round Trip (RT)
Data L1	32 KB, 2 way, WT, private, 2 cycles RT
L2	256 KB, 8 way, WB, private, 10 cycles RT
Line size	64 bytes
Coherence	MESI snoopy protocol at L2
DRAM access	$\approx$ 100 cycles RT (idle)
Node, frequency	32 nm, 2.4 GHz (can change up to 3.5 GHz)
T_{max}	105 °C
Stack DRAM Parameters	
Dies	8
Channels	4
Ranks per die	4 (1 per channel)
Banks per rank	4
I/O frequency	800 MHz
Data rate	DDR
Die capacity	4 Gb (512 MB)
Stack capacity	4 GB
Temperature Adaptive Refresh Parameters	
Discrete Mode	10 °C granularity
Continuous Mode	1 °C granularity
Guardband	2 °C granularity

To model the thermal effects in a stacked architecture we use an extension of HotSpot [91, 85]. The original HotSpot [92, 86] also models 3D-stacked architectures with multiple layers, but only allows homogeneous layers. The extension enables modeling layers with blocks that have a heterogeneous set of λ and heat capacity values. We use the grid model as opposed to the block model as it is more accurate. With this support, we start by running a processor-memory architectural simulation using SESC, McPAT and DRAMSim2, and obtain a power trace. Then, we use HotSpot to estimate the steady state temperatures induced by the power trace on a specific stack organization and TTSV placement scheme.

We run 8-threaded parallel applications from the SPLASH-2 [23], PARSEC and NAS Parallel Benchmark (NPB) [93] suites. The applications and their sizes are: Barnes (16 K particles), Cholesky (tk29.O), FFT (2^{22} points), FMM (16 K particles), LU (512x512 matrix, 16x16 blocks), Radiosity (batch), Radix (4 M integers), Raytrace (teapot), Blackscholes (sim medium), Fluidanimate (sim small),

BT (small), CG (workstation), FT (workstation), IS (workstation), LU (small), MG (workstation), and SP (small). These codes represent a diverse set.

4.5 Evaluation

In this section, we present our evaluation of Xylem. In Section 4.5.1 we show the effectiveness of TTSV placement schemes on DRAM temperature reduction. In Section 4.5.2 we show the merit of our proposed refresh schemes in the reducing the number of refreshes. We also compare two schemes that have the same TTSV count but different TTSV placement. For each TTSV placement scheme, we examine the area and routing overheads. Finally, we perform a sensitivity analysis of some parameters.

4.5.1 Impact of TTSVs on Memory Temperatures

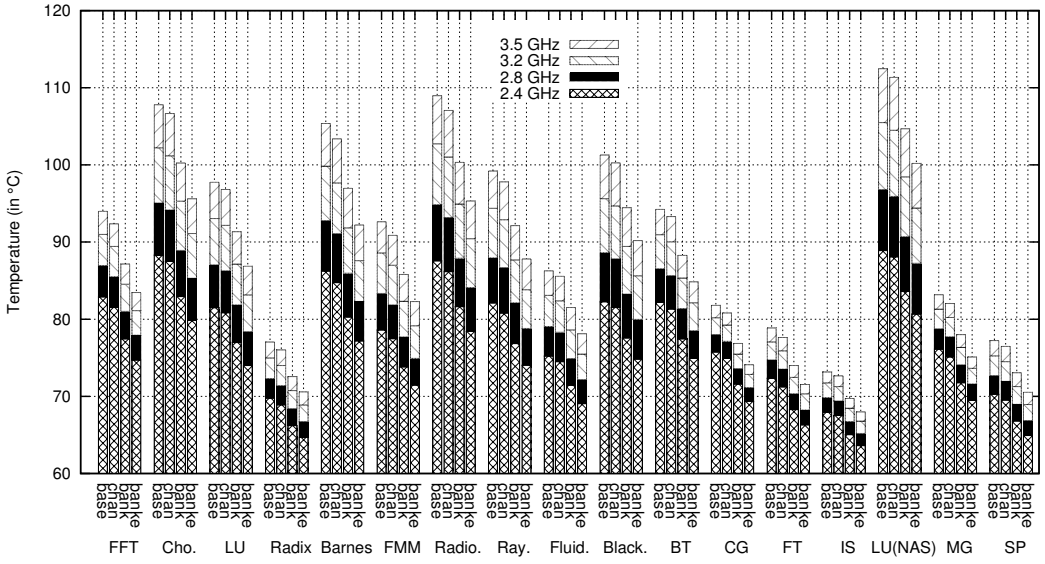


Figure 4.6: Impact of the TTSV placement schemes on the steady state temperature of the bottom most memory die.

Fig. 4.6 shows the effect of the TTSV placement schemes on the steady state temperature of the *bottom-most memory die*. The bottom-most memory die is the hottest memory die because it is closest to the processor die and farthest from the heat sink.

The figure shows four bars for each of the 17 applications, corresponding to the *base*, *chan*, *bank*, and *banke* schemes. Each bar shows the steady state temperature reached by the memory die when we run the processor at 2.4 GHz, 2.8 GHz, 3.2 GHz, and 3.5 GHz. At higher frequencies, for some applications, the figure shows the memory die temperature in excess of T_{max}. However, in a real system, processor Dynamic Thermal Management (DTM) would throttle frequencies or power-gate resources to prevent excessive temperatures.

It is easy to see that, across the board, memory temperature increases with increasing processor frequency. For example, as we go from 2.4 GHz to 3.5 GHz in *base*, the temperature increases by 6 $^{\circ}$C in FT (a memory-intensive code) and 23 $^{\circ}$C in LU (NAS) (a compute-intensive code). The temperature in *base* approaches T_{max} at 3.1 GHz for some applications, such as Cholesky, Radiosity and LU (NAS). We use 3.1 GHz as a reference frequency because the memory die temperature never exceeds T_{max}.

We now consider a given frequency and compare the temperature reached by each of the schemes. For example, take 3.1 GHz. We see that, while *chan* does not change the temperature much relative to *base*, both *bank* and *banke* schemes are highly effective at reducing the temperature.

To see the effect more clearly, Fig. 4.7 presents the data in a different way. It shows the temperature difference in $^{\circ}$C between *base* and *bank*, and between *base* and *banke* — always at 3.1 GHz. The figure shows the difference for each application and the arithmetic mean. On average, *bank* and *banke*, reduce the steady state memory die temperature by 5.5 $^{\circ}$C and 8.7 $^{\circ}$C respectively.

For DDRx [51, 52, 53] and Wide I/O [50] devices, the refresh period at 85 $^{\circ}$C is 64 ms. However, the refresh rate is doubled or quadrupled if the temperature of the DRAM die exceeds 85 $^{\circ}$C or 95 $^{\circ}$C, respectively. The impact of higher refresh rates on system energy and performance has been evaluated in [30, 94].

Going back to Figure 4.6, we see that our proposed *bank* and *banke* schemes attain temperature reduction at all frequencies. The temperature reduction allows us to lower refresh rates and hence achieve better system energy and performance.

4.5.1.1 Unaligned & Unshorted TTSVs.

Fig. 4.8 repeats the experiments of Fig. 4.6 when the TTSVs are *neither aligned nor shorted* with the dummy μbumps. We observe that simply placing TTSVs hardly offers any thermal benefits. This is in contrast to previous CAD studies [55,

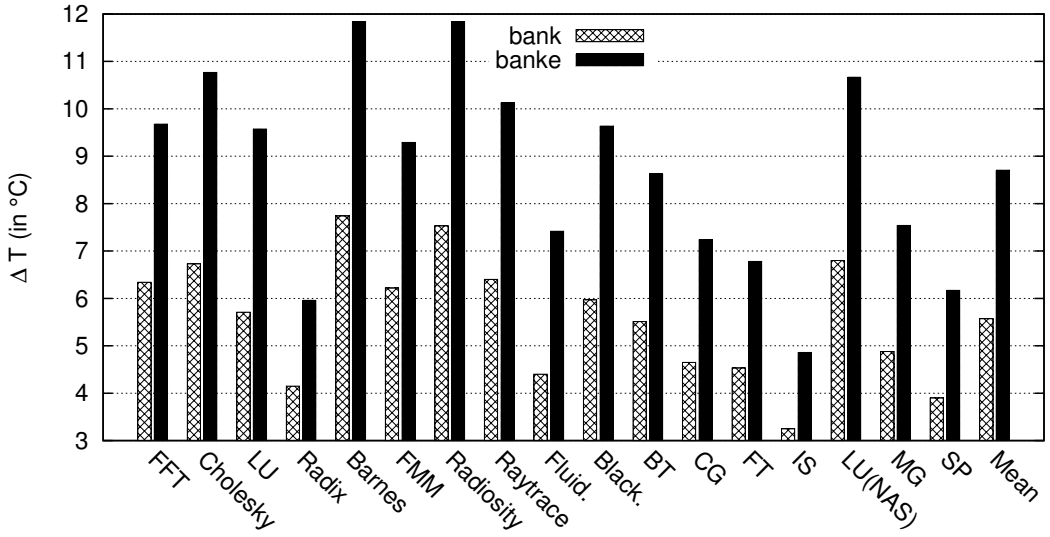


Figure 4.7: Steady-state temperature reduction over *base*.

54, 56] which suggest thermal gains with TTSV placement alone. We believe the reason is that these studies do not model the D2D layer in high fidelity. It is the combination of the TTSV placement and its alignment and shorting with dummy μbumps which offers thermal benefits.

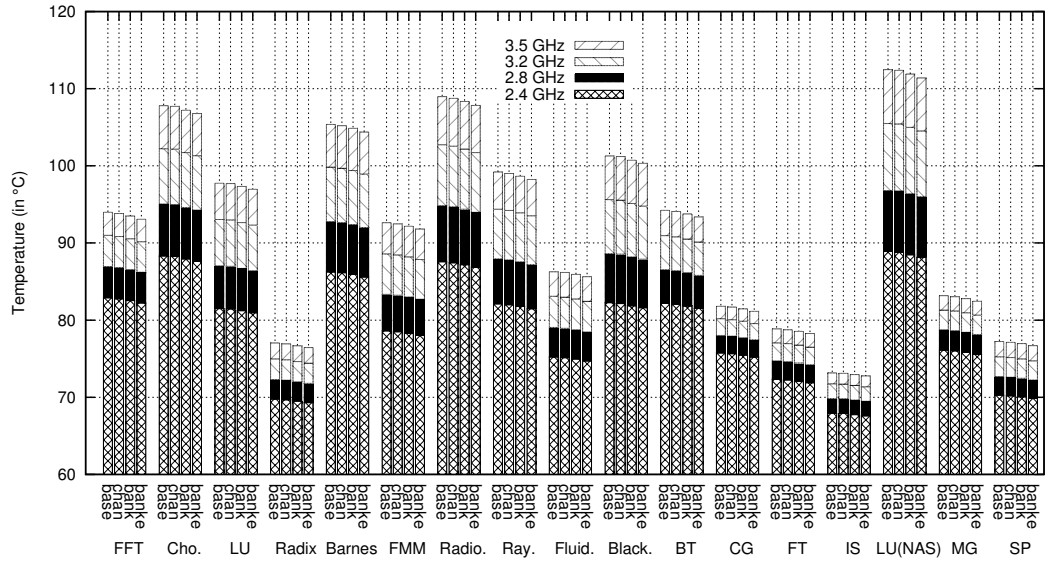


Figure 4.8: Impact of using unaligned and unshorted TTSVs on the steady state memory die temperature.

4.5.2 Effect of Refresh Schemes

In this section we evaluate the merit of our proposed refresh schemes in reducing the number of refreshes per second. Figs. 4.9, 4.10, 4.11 and 4.12 show the data for the four combinations: (i) Per Channel Refresh-Discrete Temperature Adaptation (CRDT) (ii) Per Channel Refresh-Continuous Temperature Adaptation (CRCT) (iii) Per Bank Refresh-Discrete Temperature Adaptation (BRDT) and (iv) Per Bank Refresh-Continuous Temperature Adaptation (BRCT) respectively.

In each of the four plots, the X-axis has 18 sets of bars. 17 sets of bars correspond to the 17 applications. The last set shows the average across all applications. Each set has 4 bars which correspond to the four TTSV placement schemes. The Y-axis shows the number of refreshes per second (in millions) for the entire stack.

Across all applications and on average, continuous temperature scheme is better than discrete temperature scheme, i.e. CRCT is more effective than CRDT and BRCT is more effective than BRDT. In addition, per bank refresh scheme is better than per channel refresh scheme. This is because the per bank refresh scheme can exploit the spatial variation of temperature across the stack and adjust its refresh rate to the minimum required at that temperature.

Overall, using a combination of *banke* and the best refresh scheme (BRCT), on average we reduce the number of refreshes from 70 million per second (*base* scheme with CRDT) to 11 million per second (*banke* scheme with BRCT) i.e. a reduction of 85%.

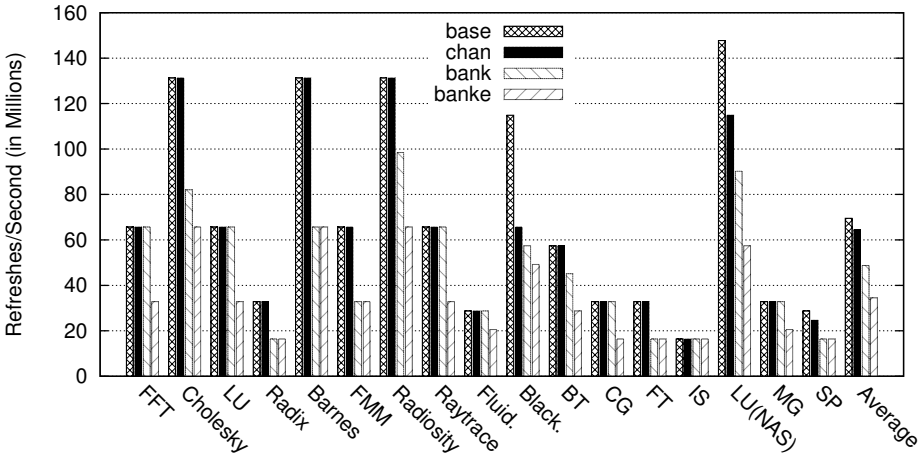


Figure 4.9: Number of refreshes for CRDT refresh scheme.

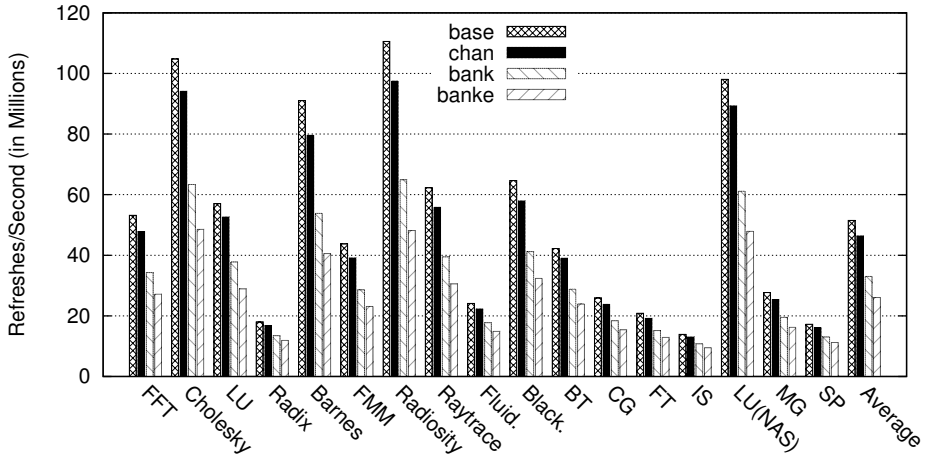


Figure 4.10: Number of refreshes for CRCT refresh scheme.

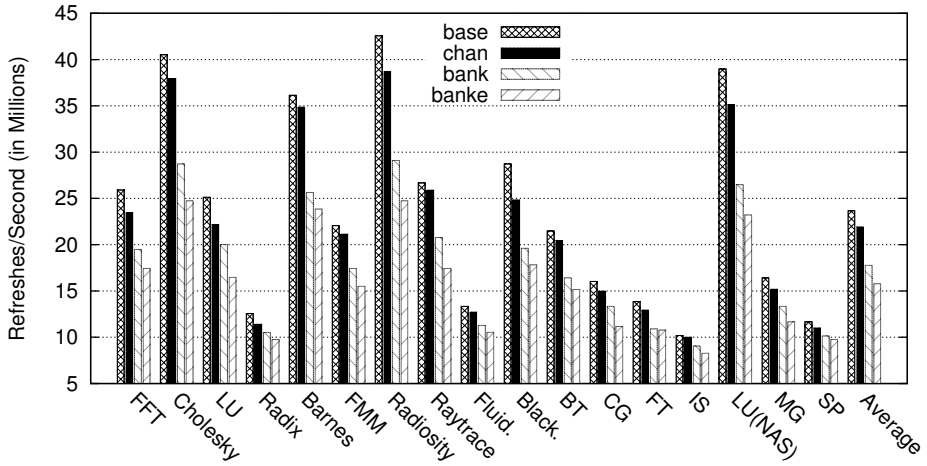


Figure 4.11: Number of refreshes for BRDT refresh scheme.

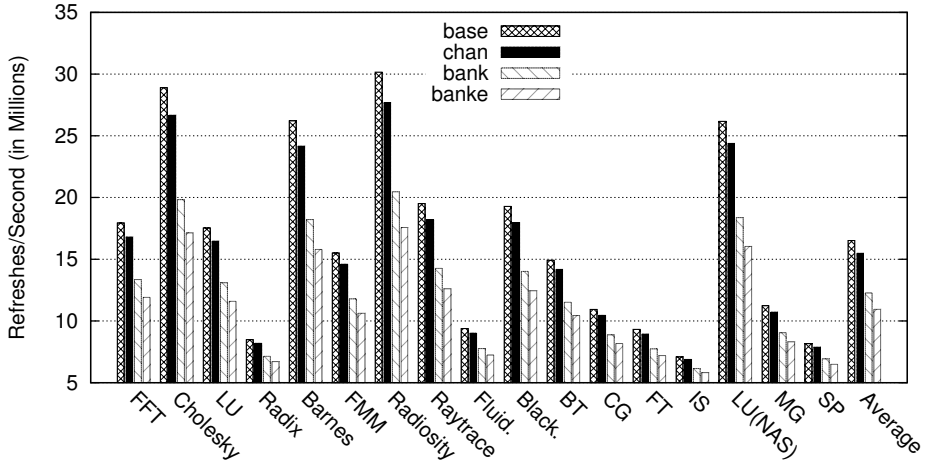


Figure 4.12: Number of refreshes for BRCT refresh scheme.

4.5.3 Comparison for a Constant Number of TTSVs

The TTSV placement schemes evaluated so far differ in the total number of TTSVs they use. We now consider two schemes listed in Table 4.3, namely *chan* and *peri* (for periphery). These schemes have the same TTSV count (10) but different TTSV placement. *peri* is like *chan* except that it removes the 4 TTSVs from the middle left and right of the die and places them in the top and bottom edges of the die, one in the middle of each channel's side. By comparing the two schemes, we are eliminating the effect of having more TTSVs and focusing on the impact of where we place the TTSVs.

Fig. 4.13 shows the effect of the *chan* and *peri* TTSV placement schemes on the steady-state temperature of the memory die. The figure is organized like Fig. 4.6. Across all applications, we see that *peri* reduces temperatures by 1-3 °C. Hence, it matters where we place the TTSVs.

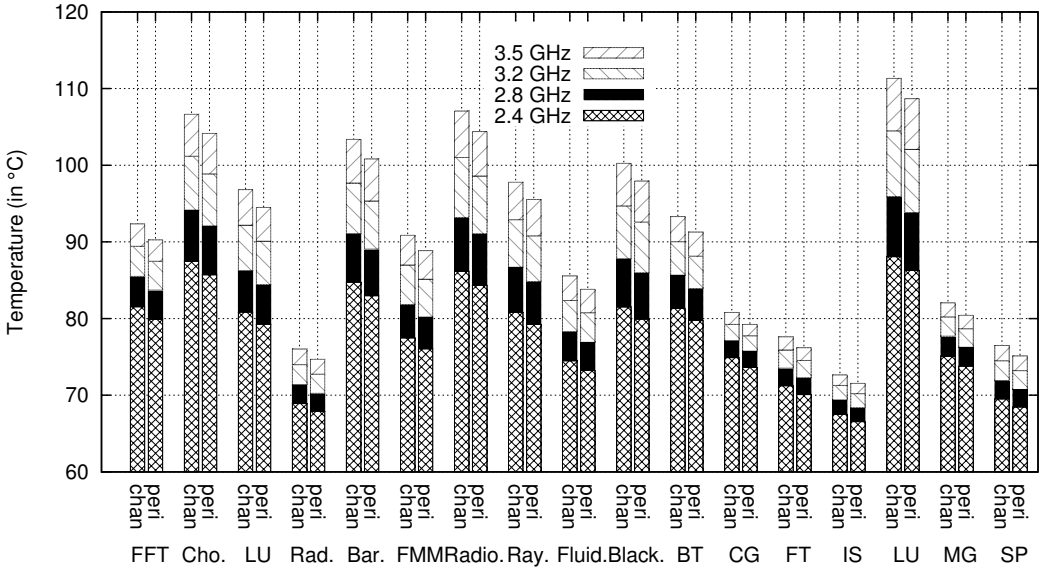


Figure 4.13: Impact of TTSV placement on memory die temperature for constant TTSV count.

4.5.4 TTSV Area & Routing Overheads

Based on the TTSV parameters of Section 4.4, the area of one TTSV plus its KOZ is 0.0144 mm^2. Given the total number of TTSVs in each of the placement schemes described in Table 4.3, we compute the total area taken by all the TTSVs in a chip as 0.1440 mm^2, 0.4032 mm^2, and 0.5184 mm^2 in *chan*, *bank*, and *banke*,

respectively. Compared to the 64.34 mm^2 die area reported by Samsung in the Wide I/O prototype [63], this is an insignificant 0.22%, 0.62%, and 0.80% of the chip area, respectively. In the best case, TTSVs can be placed in the dead spaces of the peripheral logic, resulting in no area overheads. Moreover, since the TTSVs are passive, they do not have any energy overhead. Also, as shown in Fig. 4.5, TTSVs are not present in the frontside metal layers and, hence, do not cause any routing congestion or overheads there.

4.5.5 Sensitivity Analysis

4.5.5.1 Effect of Die Thickness

For a constant TSV aspect ratio, die thinning is attractive to increase the TSV interconnect density. However, Emma et al. [62] observed that die thinning worsens chip temperatures. Therefore, there is a trade-off between TSV interconnect density and chip temperatures. We made the same observation. Fig. 4.14 shows the effect of die thickness on memory temperature, averaged over all applications, at 2.4 GHz. The figure shows 3 sets of bars, each corresponding to a different die thickness. Each set has 4 bars, corresponding to the 4 TTSV placement schemes. As expected, memory temperatures become worse with die thinning.

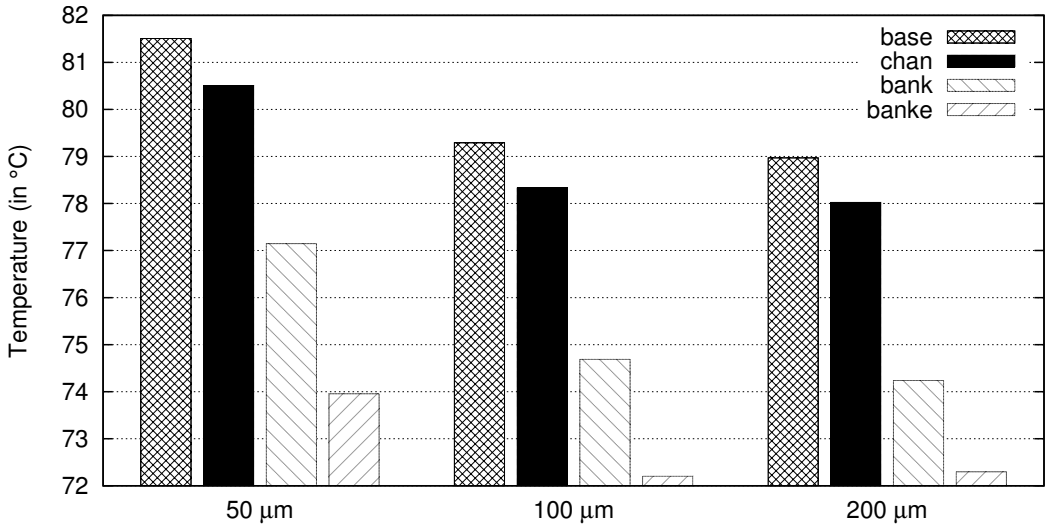


Figure 4.14: Impact of die thickness on temperature.

4.5.5.2 Effect of Number of Memory Dies

Intuitively, increasing the number of memory dies increases the transistor density and hence the power density in the stack. It also increases the distance of the memory die from the heat sink. Therefore, we expect the memory temperatures to increase. Fig. 4.15 shows the effect of number of memory dies in the stack on memory temperature, averaged over all applications, at 2.4 GHz. The figure shows 3 sets of bars, each corresponding to a different number of memory dies in the stack. Each set has 4 bars, corresponding to the 4 TTSV placement schemes. As expected, the memory temperatures become worse with an increasing number of memory dies.

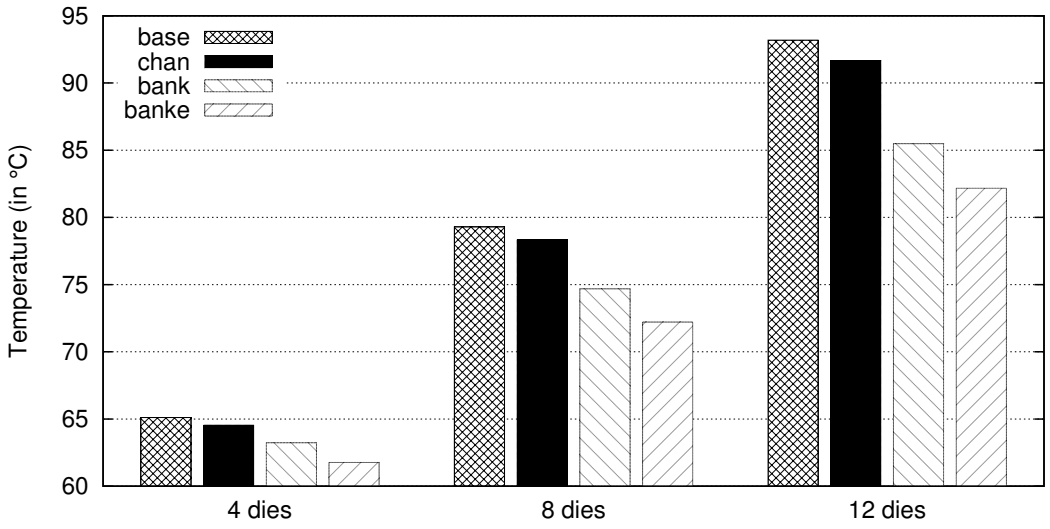


Figure 4.15: Impact of the number of dies on temperature.

4.6 Related Work

The CAD community has examined algorithms to place TTSVs and to minimize their count [55, 54, 56, 57, 58]. However, we find that these works either ignore some physical implementation issues of TSVs, or use very aggressive D2D layer parameters. In either case, they do not fully consider the D2D layer resistance and, hence, TTSV placements alone appear to offer thermal savings.

Goplen and Sapatnekar [54, 56] reserve certain regions, called thermal via regions, for TTSV placement. These regions are uniformly placed throughout the

chip and occupy 10% of the chip area. Their algorithm determines the number of TTSVs in each of these regions to minimize the overall TTSV count. The algorithm requires knowledge of all power densities in the stack. The approach and evaluation has a few limitations: (1) the algorithm is applicable only for standard cell implementations, (2) one should not place TTSVs uniformly in a memory die, since they would disrupt the regular DRAM array layout (TTSVs should be present only in the peripheral region), (3) in general, the memory vendor does not have information about the processor power densities or hotspots needed by the algorithm, and (4) the evaluation uses a D2D layer thickness of 0.7 μm. This value is over 20x lower than the state of the art and, therefore, the effects of the D2D layer resistance are not considered. Finally, an area overhead of 10% is substantial.

Cong and Zhang [55] propose a heuristic algorithm to minimize TTSV count in a more generic layout of blocks in a die. The algorithm inserts TTSVs in the white spaces between cells and macros. However, the algorithm requires knowledge of all the layers of the stack, which is not available to a memory vendor. In addition, the TTSV is assumed to directly connect to the metal layers of the adjacent die. This assumption ignores the physical implementation of a TSV and the presence of the D2D layer.

Ganeshpure and Kundu [95] propose Heat Pipes as a heat transfer mechanism, where placing TTSVs directly at the hotspots is difficult due to wiring congestion. Chen et al. [73] propose an algorithm for TSV placement with the goal of mitigating the lateral thermal blockage effects of TSVs.

Many architectural studies have looked at implications of 3D integration for logic and memory, from both the performance and thermal standpoint. Recently, Emma et al. [62] proposed different modes of operation for processor-on-processor stacking. They also analyze the impact of die thickness and hotspot offset on temperature. They do not propose temperature reducing techniques.

Puttaswamy and Loh [96, 97] propose and analyze techniques for thermal management for a 3D stacked processor (not a generic 3D processor-memory organization). In Thermal Herding, they propose moving the hottest datapaths (16 LSBs) closest to the heat sink. Extending their proposal to a generic 3D processor-memory stack would imply moving the processor die closest to the heat sink, resulting in our 'processor-on-top' configuration. In Sec. 4.2, we explained the manufacturing limitations of this configuration and hence it was not evaluated. In addition, they do not discuss the issue of thermal resistance, TTSVs, the D2D

layer or μbumps.

Black et al. [47] and Loh [48] look at the performance benefits and thermal challenges of a 3D processor-memory stacked architecture. Black et al.'s thermal analysis is for a 2-die stack — i.e., a processor and a memory die in a f2f configuration. They discuss the impact of the D2D layer and the frontside metal layer conductivity on temperature. Loh considers a multi-layered stack in a f2b configuration. However, both evaluations assume a 'processor-on-top' configuration. In addition, Loh assumes that the D2D layer has a λ = 100 W/(m·K) (a quarter of bulk copper) and a thickness of 2 μm. In practice, according to experimental measurements by Matsumoto et al. [64], the D2D has a λ that is $65x$ smaller, and a thickness that is about $10x$ higher. As a result, Loh observes only a 10 $^{\circ}$C temperature impact for a 16 layer memory stack.

There has been a lot of interest in techniques to reduce refresh energy of DRAMs and embedded DRAMs (eDRAM). While Mosaic [49] and RAIDR [40] reduce refresh energy by taking advantage of variation in retention time, Smart Refresh [30] and Refrint [45] leverage DRAM access patterns for the same. In [30], Ghosh and Lee also consider the impact of higher refresh rates in DRAMs, due to higher temperatures in a 3D stack. The performance impact due to higher temperatures in stacked DRAM is also studied by Loi et al. [94]. Hi-ECC [3] and HEAR [98] decrease the number of refreshes and employ strong Error Correction Codes (ECC) to compensate for data errors. In [99], the authors leverage retention time variation for critical data placement to reduce power consumption. While they touch on the influence of temperature on DRAM refresh, detailed studies on the impact of temperature with specific planar/stacked memory organizations are missing.

4.7 Summary

In 3D architectures that stack processor and DRAM dies, thermal considerations are paramount. In particular DRAM dies experience both high temperatures and spatial variations in temperature. However, there is little work on thermal management for 3D memory dies.

This proposal has focused on thermal and refresh management for 3D processor-memory stacks. Our scheme, called *Xylem* involved: (1) placing the TTSVs on die subject to architecture and implementation constraints, (2) aligning and short-

ing the TTSVs with μbumps to avoid paths of high thermal resistance and (3) new DRAM refresh schemes to exploit spatial variation in temperature.

We proposed three TTSV placement schemes: two generic ones (*chan* and *bank*) and a custom one (*banke*). We evaluated them using simulations of an 8-core die running at 3.1 GHz and 8 DRAM dies on top. We ran 17 applications. *bank* and *banke* reduced the peak temperature of the DRAM stack by an average of 5.5 $^{\circ}\mathrm{C}$ and 8.7 $^{\circ}\mathrm{C}$ respectively. Combined with the best DRAM refresh scheme we reduced the number of refreshes by an average of 85%. We also showed that TTSV placement is more important than TTSV count.

CHAPTER 5

CONCLUSION

An effective approach to reduce the static energy consumption of large on-chip memories is to use a low-leakage technology such as embedded DRAM (eDRAM). Unfortunately, eDRAM, being a dynamic memory, requires periodic refresh, which ends up consuming substantial energy for large last-level caches.

In upcoming architectures that stack a processor die and multiple DRAM dies, DRAM dies experience higher temperatures. Elevated temperatures increase the periodic refresh requirement of DRAM, also a dynamic memory, which increases its energy and hurts processor performance. In this thesis, we proposed and evaluated techniques for refresh reduction in dynamic memories.

In *Refrint* our goal was to refresh only the data that will be used in the near future, and only if the data has not been recently accessed (and automatically refreshed). We introduced the Refrint algorithms and microarchitecture.

Our results showed that Refrint is very effective. On average, an eDRAM-based memory hierarchy without Refrint consumed 56% of the energy of a conventional SRAM-based memory hierarchy. However, it increased the execution time of the applications by 25%. On the other hand, an eDRAM-based memory hierarchy with Refrint only consumed 30% of the energy of the conventional SRAM-based memory hierarchy. In addition, it only increased the execution time of the applications by 6%. In this environment, the contribution of refreshes in the energy remaining was negligible.

In *Mosaic* we presented a new, high-level model of the retention times in large on-chip eDRAM modules. This model found that the retention properties of cells in large eDRAM modules exhibit spatial correlation. Based on the model, we proposed a simple architecture to exploit this correlation to save much of the refresh energy at low cost.

We evaluated Mosaic on a 16 core chip multiprocessor running 16-threaded parallel applications. We found that Mosaic is both inexpensive and very effective. An eDRAM L3 cache augmented with Mosaic increased its area by 2% and

reduced its refresh energy by 20 times. With Mosaic, we saved 43% of the total energy in a large on-chip eDRAM L3 cache, and got very close to the lower bound in refresh energy.

In Xylem we focused on thermal and refresh management for 3D processor-memory stacks. We proposed using Thermal Through Silicon Vias (TTSVs), to reduce the temperature of the DRAM dies in the stack. Our scheme involved: (1) placing the TTSVs on die subject to architecture and implementation constraints, (2) aligning and shorting the TTSVs with μbumps to avoid paths of high thermal resistance and (3) new DRAM refresh schemes to exploit spatial variation in temperature.

We proposed three TTSV placement schemes: two generic ones (*chan* and *bank*) and a custom one (*banke*). We evaluated them using simulations of an 8-core die running at 3.1 GHz and 8 DRAM dies on top. We ran 17 applications. *bank* and *banke* reduced the peak temperature of the DRAM stack by an average of 5.5 $^{\circ}\mathrm{C}$ and 8.7 $^{\circ}\mathrm{C}$ respectively. Combined with the best DRAM refresh scheme we reduced the number of refreshes by an average of 85%. We also showed that TTSV placement is more important than TTSV count.

REFERENCES

[1] D. F. Wendel, R. N. Kalla, J. D. Warnock, R. Cargnoni, S. G. Chu, J. G. Clabes, D. Dreps, D. Hrusecky, J. Friedrich, M. S. Islam, J. A. Kahle, J. Leenstra, G. Mittal, J. Paredes, J. Pille, P. J. Restle, B. Sinharoy, G. Smith, W. J. Starke, S. Taylor, J. V. Norstrand, S. Weitzel, P. G. Williams, and V. V. Zyuban, "POWER7: A Highly Parallel, Scalable Multi-Core High End Server Processor," *IEEE Journal of Solid State Circuits*, Jan. 2011.

[2] S. S. Iyer, J. E. B. Jr., P. C. Parries, J. P. Norum, J. P. Rice, L. R. Logan, and D. Hoyniak, "Embedded DRAM: Technology Platform for the Blue Gene/L Chip," *IBM Journal of Research and Development*, Mar. 2005.

[3] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-L. Lu, "Reducing Cache Power with Low-cost, Multi-bit Error-Correcting Codes," in *International Symposium on Computer Architecture*, Jun. 2010.

[4] K. Banerjee, S. Souri, P. Kapur, and K. Saraswat, "3-D ICs: A Novel Chip Design for Improving Deep-Submicrometer Interconnect Performance and Systems-on-Chip Integration," *Proceedings of the IEEE*, May 2001.

[5] S. Borkar, "3D Integration for Energy Efficient System Design," in *IEEE Design Automation Conference*, Jun. 2011.

[6] T.-Y. Chiang, S. Souri, C. O. Chui, and K. Saraswat, "Thermal Analysis of Heterogeneous 3D ICs with Various Integration Scenarios," in *International Electron Devices Meeting*, Dec. 2001.

[7] M. Koyanagi, H. Kurino, K.-W. Lee, K. Sakuma, N. Miyakawa, and H. Itani, "Future System-on-Silicon LSI Chips," *IEEE Micro*, Jul. 1998.

[8] W. Kong, P. Parries, G. Wang, and S. Iyer, "Analysis of Retention Time Distribution of Embedded DRAM - A New Method to Characterize Across-Chip Threshold Voltage Variation," in *IEEE International Test Conference*, Oct. 2008.

[9] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multi-threaded Sparc Processor," *IEEE Micro*, Mar.-Apr. 2005.

[10] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," *Proceedings of the IEEE*, Feb. 2010.

[11] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy Caches: Simple Techniques for Reducing Leakage Power," in *International Symposium on Computer Architecture*, May 2002.

[12] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power," in *International Symposium on Computer Architecture*, Jun. 2001.

[13] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. Vijaykumar, "Gated-V_{dd}: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," in *International Symposium on Low Power Electronics and Design*, Jul. 2000.

[14] S. Yang, M. Powell, B. Falsafi, K. Roy, and T. Vijaykumar, "An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches," in *IEEE International Symposium on High Performance Computer Architecture*, Jan. 2001.

[15] S.-H. Yang, M. Powell, B. Falsafi, and T. Vijaykumar, "Exploiting Choice in Resizable Cache Design to Optimize Deep-Submicron Processor Energy-Delay," in *IEEE International Symposium on High Performance Computer Architecture*, Feb. 2002.

[16] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte, "Adaptive Mode Control: A Static-Power-Efficient Cache Design," in *IEEE International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2001.

[17] S. Petit, J. Sahuquillo, J. M. Such, and D. Kaeli, "Exploiting Temporal Locality in Drowsy Cache Policies," in *Computing Frontiers Conference*, May 2005.

[18] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. Jouppi, "CACTI 5.1. Technical Report," Hewlett Packard Labs, Tech. Rep., Apr. 2008.

[19] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, "SESC simulator," Jan. 2005. [Online]. Available: http://sesc.sourceforge.net

[20] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *IEEE International Symposium on Microarchitecture*, Dec. 2009.

[21] J. Barth, W. Reohr, P. Parries, G. Fredeman, J. Golz, S. Schuster, R. Matick, H. Hunter, C. Tanner, J. Harig, K. Hoki, B. Khan, J. Griesemer, R. Havreluk, K. Yanagisawa, T. Kirihata, and S. Iyer, "A 500 MHz Random Cycle 1.5ns Latency, SOI Embedded DRAM Macro Featuring a 3T Micro Sense Amplifier," *IEEE International Solid State Circuits Conference*, Feb. 2008.

[22] K. C. Chun, W. Zhang, P. Jain, and C. Kim, "A 700 MHz 2T1C Embedded DRAM Macro in a Generic Logic Process with No Boosted Supplies," in *International Solid-State Circuits Conference*, Feb. 2011.

[23] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *IEEE International Symposium on Computer Architecture*, Jun. 1995.

[24] J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies," in *International Symposium on Computer Architecture*, Jun. 2008.

[25] R. E. Matick and S. Schuster, "Logic-based eDRAM: Origins and Rationale for use," *IBM Journal of Research and Development*, Jan. 2005.

[26] X. Liang, R. Canal, G.-Y. Wei, and D. Brooks, "Process Variation Tolerant 3T1D-Based Cache Architectures," in *IEEE International Symposium on Microarchitecture*, Dec. 2007.

[27] Z. Hu, P. Juang, P. Diodato, S. Kaxiras, K. Skadron, M. Martonosi, and D. W. Clark, "Managing Leakage for Transient Data: Decay and Quasi-Static 4T Memory Cells," in *International Symposium on Low Power Electronics and Design*, Aug. 2002.

[28] A. Valero, J. Sahuquillo, S. Petit, V. Lorente, R. Canal, P. López, and J. Duato, "An Hybrid eDRAM/SRAM Macrocell to Implement First-Level Data Caches," in *IEEE International Symposium on Microarchitecture*, Dec. 2009.

[29] R. K. Venkatesan, S. Herr, and E. Rotenberg, "Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM," in *International Symposium on High Performance Computer Architecture*, Feb. 2006.

[30] M. Ghosh and H.-H. S. Lee, "Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs," in *IEEE International Symposium on Microarchitecture*, Dec. 2007.

[31] P. Emma, W. Reohr, and M. Meterelliyoz, "Rethinking Refresh: Increasing Availability and Reducing Power in DRAM for Cache Applications," *IEEE Micro*, Nov.-Dec. 2008.

[32] T. Hamamoto, S. Sugiura, and S. Sawada, "On the Retention Time Distribution of Dynamic Random Access Memory (DRAM)," *IEEE Transactions on Electron Devices*, Jun. 1998.

[33] K. Kim and J. Lee, "A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs," *IEEE Electron Device Letters*, Aug. 2009.

[34] M. Horiguchi, "Redundancy Techniques for High-Density DRAMs," in *IEEE International Conference on Innovative Systems in Silicon*, Oct. 1997.

[35] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos, "Modeling Within-Die Spatial Correlation Effects for Process-Design Co-Optimization," in *International Symposium on Quality of Electronic Design*, Mar. 2005.

[36] M. Orshansky, L. Milor, and C. Hu, "Characterization of Spatial Intrafield Gate CD Variability, Its Impact on Circuit Performance, and Spatial Mask-Level Correction," *IEEE Transactions on Semiconductor Manufacturing*, Feb. 2004.

[37] B. Stine, D. Boning, and J. Chung, "Analysis and Decomposition of Spatial Variation in Integrated Circuit Processes and Devices," *IEEE Transactions on Semiconductor Manufacturing*, Feb. 1997.

[38] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects," *IEEE Transactions on Semiconductor Manufacturing*, Feb. 2008.

[39] T. Karnik, S. Borkar, and V. De, "Probabilistic and Variation-Tolerant Design: Key to Continued Moore's Law," in *IEEE TAU Workshop*, Feb. 2004.

[40] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *International Symposium on Computer Architecture*, Jun. 2012.

[41] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *International Symposium on Computer Architecture*, Jun. 2013.

[42] N. Weste, K. Eshraghian, and M. Smith, *Principles of CMOS VLSI Design: A Systems Perspective*. Prentice Hall, 2000.

[43] J.-H. Choi, K.-S. Noh, and Y.-H. Seo, "Methods of Operating DRAM Devices having Adjustable Internal Refresh Cycles that vary in Response to On-Chip Temperature Changes," Patent US 8 218 137, Jul., 2012.

[44] "The R Project for Statistical Computing." [Online]. Available: http://www.r-project.org/

[45] A. Agrawal, P. Jain, A. Ansari, and J. Torrellas, "Refrint: Intelligent Refresh to Minimize Power in On-Chip Multiprocessor Cache Hierarchies," in *International Symposium on High Performance Computer Architecture*, Feb. 2013.

[46] M.-T. Chang, P. Rosenfeld, S.-L. Lu, and B. Jacob, "Technology Comparison for Large Last-Level Caches (L3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM," in *IEEE International Symposium on High Performance Computer Architecture*, Feb. 2013.

[47] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. Loh, D. McCauley, P. Morrow, D. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die Stacking (3D) Microarchitecture," in *IEEE International Symposium on Microarchitecture*, Dec. 2006.

[48] G. Loh, "3D-Stacked Memory Architectures for Multi-core Processors," in *International Symposium on Computer Architecture*, Jun. 2008.

[49] A. Agrawal, A. Ansari, and J. Torrellas, "Mosaic: Exploiting the Spatial Locality of Process Variation to Reduce Refresh Energy in On-Chip eDRAM Modules," in *International Symposium on High Performance Computer Architecture*, Feb. 2014.

[50] "Wide I/O SDR Standard," 2011. [Online]. Available: http://www.jedec.org/standards-documents/results/jesd229

[51] "DDR2 SDRAM Standard," 2009. [Online]. Available: http://www.jedec.org/standards-documents/docs/jesd-79-2e

[52] "DDR3 SDRAM Standard," 2012. [Online]. Available: http://www.jedec.org/standards-documents/docs/jesd-79-3d

[53] "Low Power DDR3 SDRAM Standard," 2013. [Online]. Available: http://www.jedec.org/standards-documents/results/jesd209-3

[54] B. Goplen and S. S. Sapatnekar, "Thermal Via Placement in 3D ICs," in *International Symposium on Physical Design*, Apr. 2005.

[55] J. Cong and Y. Zhang, "Thermal Via Planning for 3-D ICs," in *IEEE International Conference on Computer-Aided Design*, Nov. 2005.

[56] B. Goplen and S. S. Sapatnekar, "Placement of Thermal Vias in 3-D ICs Using Various Thermal Objectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Apr. 2006.

[57] S. Singh and C. S. Tan, "Impact of Thermal Through Silicon Via (TTSV) on the Temperature Profile of Multi-Layer 3-D Device Stack," in *IEEE International Conference on 3D System Integration*, Sep. 2009.

[58] G. Van der Plas, P. Limaye, A. Mercha, H. Oprins, C. Torregiani, S. Thijs, D. Linten, M. Stucchi, K. Guruprasad, D. Velenis, D. Shinichi, V. Cherman, B. Vandevelde, V. Simons, I. De Wolf, R. Labie, D. Perry, S. Bronckers, N. Minas, M. Cupac, W. Ruythooren, J. Van Olmen, A. Phommahaxay, M. de Potter de ten Broeck, A. Opdebeeck, M. Rakowski, B. De Wachter, M. Dehan, M. Nelis, R. Agarwal, W. Dehaene, Y. Travaly, P. Marchal, and E. Beyne, "Design Issues and Considerations for Low-Cost 3D TSV IC Technology," in *IEEE International Solid State Circuits Conference*, Jan. 2010.

[59] E.-H. Chen, T.-C. Hsu, C.-H. Lin, P.-J. Tzeng, C.-C. Wang, S.-C. Chen, J.-C. Chen, C.-C. Chen, Y.-C. Hsin, P.-C. Chang, Y.-H. Chang, S.-C. Chen, Y. ming Lin, S.-C. Liao, and T.-K. Ku, "Fine-pitch Backside Via-last TSV Process with Optimization on Temporary Glue and Bonding Conditions," in *IEEE Electronic Components and Technology Conference*, May 2013.

[60] K. Chen and C. Tan, "Integration Schemes and Enabling Technologies for Three-Dimensional Integrated Circuits," *IET Computers & Digital Techniques*, May 2011.

[61] S. C. Johnson, "Via First, Middle, Last, or After?" *3D Packaging Newsletter on 3D IC, TSV, WLP & Embedded Technologies*, Dec. 2009. [Online]. Available: http://www.i-micronews.com/upload\%5Cnewsletter\%5C3DNov09.pdf

[62] P. Emma, A. Buyuktosunoglu, M. Healy, K. Kailas, V. Puente, R. Yu, A. Hartstein, P. Bose, and J. Moreno., "3D Stacking of High-Performance Processors," in *IEEE International Symposium on High-Performance Computer Architecture*, Feb. 2014.

[63] J.-S. Kim, C. S. Oh, H. Lee, D. Lee, H.-R. Hwang, S. Hwang, B. Na, J. Moon, J.-G. Kim, H. Park, J.-W. Ryu, K. Park, S.-K. Kang, S.-Y. Kim, H. Kim, J.-M. Bang, H. Cho, M. Jang, C. Han, J.-B. Lee, K. Kyung, J.-S. Choi, and Y.-H. Jun, "A 1.2V 12.8GB/s 2Gb Mobile Wide-I/O DRAM with 4x128 I/Os Using TSV-Based Stacking," in *IEEE International Solid State Circuits Conference*, Feb. 2011.

[64] K. Matsumoto, S. Ibaraki, K. Sakuma, K. Sueoka, H. Kikuchi, Y. Orii, and F. Yamada, "Thermal Resistance Evaluation of a Three-dimensional (3D) Chip Stack," in *Electronics Packaging Technology Conference*, Dec. 2010.

[65] "Hybrid Memory Cube Consortium." [Online]. Available: http://hybridmemorycube.org/

[66] "High Bandwidth Memory (HBM) Standard," 2013. [Online]. Available: http://www.jedec.org/standards-documents/results/jesd235

[67] "Joint Electron Device Engineering Council (JEDEC)." [Online]. Available: http://www.jedec.org/

[68] [Online]. Available: http://www.cadence.com/Community/blogs/ii/archive/2013/08/06/wide-i-o-2-hybrid-memory-cube-hmc-memory-models-advance-3d-ic-standards.aspx

[69] [Online]. Available: http://www.xbitlabs.com/news/memory/display/20120606201618_JEDEC_Considers_Wide_I_O_Wide_I_O_2_Interfaces_for_PCs_and_Servers.html

[70] C.-K. Kim, J.-G. Lee, Y.-H. Jun, C.-G. Lee, and B.-S. Kong, "CMOS Temperature Sensor with Ring Oscillator for Mobile DRAM Self-Refresh Control," *IEEE International Symposium on Circuits and Systems*, May 2008.

[71] E. Samson, S. Machiroutu, J.-Y. Chang, I. Santos, J. Hermerding, A. Dani, R. Prasher, and D.W.Song, "Interface Material Selection and a Thermal Management Technique in Second-Generation Platforms Built on Intel Centrino Mobile Technology," in *Intel Technology Journal*, Feb. 2005.

[72] M. Shevgoor, J.-S. Kim, N. Chatterjee, R. Balasubramonian, A. Davis, and A. N. Udipi, "Quantifying the Relationship Between the Power Delivery Network and Architectural Policies in a 3D-stacked Memory Device," in *IEEE International Symposium on Microarchitecture*, Dec. 2013.

[73] Y. Chen, E. Kursun, D. Motschman, C. Johnson, and Y. Xie, "Analysis and Mitigation of Lateral Thermal Blockage Effect of Through-Silicon-Via in 3D IC Designs," in *International Symposium on Low Power Electronics and Design*, Aug. 2011.

[74] K. Athikulwongse, A. Chakraborty, J.-S. Yang, D. Pan, and S.-K. Lim, "Stress-Driven 3D-IC Placement with TSV Keep-Out Zone and Regularity Study," in *IEEE International Conference on Computer-Aided Design*, Nov. 2010.

[75] T. Song, C. Liu, Y. Peng, and S. K. Lim, "Full-Chip Multiple TSV-to-TSV Coupling Extraction and Optimization in 3D ICs," in *IEEE Design Automation Conference*, May-Jun. 2013.

[76] R. Golla and P. Jordan, "T4: A Highly Threaded Server-on-a-Chip with Native Support for Heterogeneous Computing," in *Hot Chips: A Symposium on High Performance Chips*, Aug. 2011.

[77] R. Kalla, "POWER7: IBMs Next Generation POWER Microprocessor," in *Hot Chips: A Symposium on High Performance Chips*, Aug. 2009.

[78] J. Stuecheli, "Next Generation POWER Microprocessor," in *Hot Chips: A Symposium on High Performance Chips*, Aug. 2013.

[79] S. Turullols and R. Sivaramakrishnan, "16-core SPARC T5 CMT Processor with Glueless 1-hop Scaling to 8-sockets," in *Hot Chips: A Symposium on High Performance Chips*, Aug. 2012.

[80] S. Undy, "Poulson: An 8 Core 32 nm Next Generation Intel Itanium Processor," in *Hot Chips: A Symposium on High Performance Chips*, Aug. 2011.

[81] S. White, "High Performance Power-Efficient x86-64 Server & Desktop Processors: using Bulldozer Core," in *Hot Chips: A Symposium on High Performance Chips*, Aug. 2011.

[82] D. Noice and V. Gerousis, "Physical Design Implementation for 3D IC: Methodology and Tools," *International Symposium on Physical Design*, Mar. 2010, Invited talk. [Online]. Available: http://www.ispd.cc/slides/slides10/4_02.pdf

[83] "Micron DDR4 SDRAM." [Online]. Available: http://www.micron.com/products/dram/ddr4-sdram

[84] "International Technology Roadmap for Semiconductors (ITRS)," 2012. [Online]. Available: http://www.itrs.net/Links/2012ITRS/2012Tables/Interconnect_2012Tables.xlsx

[85] J. Meng, K. Kawakami, and A. Coskun, "Optimizing Energy Efficiency of 3-D Multicore Systems with Stacked DRAM under Power and Thermal Constraints," in *IEEE Design Automation Conference*, Jun. 2012.

[86] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, May 2006.

[87] "DRAMSim2." [Online]. Available: http://www.eng.umd.edu/~blj/dramsim/

[88] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *Computer Architecture Letters*, Jan.-Jun. 2011.

[89] "ArchFP." [Online]. Available: http://lava.cs.virginia.edu/archfp/

[90] G. Faust, R. Zhang, K. Skadron, M. Stan, and B. Meyer, "ArchFP: Rapid Prototyping of pre-RTL Floorplans," in *IEEE International Conference on VLSI and System-on-Chip*, Oct. 2012.

[91] "HotSpot 3D Extension." [Online]. Available: http://lava.cs.virginia.edu/HotSpot/links.htm

[92] "HotSpot." [Online]. Available: http://lava.cs.virginia.edu/HotSpot/index.htm

[93] "NAS Parallel Benchmarks." [Online]. Available: http://www.nas.nasa.gov/publications/npb.html

[94] G. Loi, B. Agrawal, N. Srivastava, S.-C. Lin, T. Sherwood, and K. Banerjee, "A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy," in *IEEE Design Automation Conference*, Jul. 2006.

[95] K. Ganeshpure and S. Kundu, "Reducing Temperature Variation in 3D Integrated Circuits Using Heat Pipes," in *IEEE Symposium on VLSI*, Aug. 2012.

[96] K. Puttaswamy and G. H. Loh, "Thermal Analysis of a 3D Die-Stacked High-Performance Microprocessor," in *Proceedings of the Great Lakes Symposium on VLSI*, Apr.-May 2006.

[97] K. Puttaswamy and G. Loh, "Thermal Herding: Microarchitecture Techniques for Controlling Hotspots in High-Performance 3D-Integrated Processors," in *IEEE International Symposium on High Performance Computer Architecture*, Feb. 2007.

[98] Y.-C. Yu, C.-S. Hou, L.-J. Chang, J.-F. Li, C.-Y. Lo, D.-M. Kwai, Y.-F. Chou, and C.-W. Wu, "A Hybrid ECC and Redundancy Technique for Reducing Refresh Power of DRAMs," in *IEEE VLSI Test Symposium*, Apr.-May 2013.

[99] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM Refresh-power through Critical Data Partitioning," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2011.