EXPLOITING ON-CHIP MEMORY CONCURRENCY IN 3D MANYCORE ARCHITECTURES

A Dissertation
Presented to
The Academic Faculty

By

Syed Minhaj Hassan

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
December 2016

EXPLOITING ON-CHIP MEMORY CONCURRENCY IN 3D MANYCORE ARCHITECTURES

Approved by:

Dr. Sudhakar Yalamanchili, Advisor *Joseph M. Pettit Professor, School of ECE* *Georgia Institute of Technology*	Dr. Saibal Mukhopadhyay *Associate Professor, School of ECE* *Georgia Institute of Technology*
Dr. Tushar Krishna *Assistant Professor, School of ECE* *Georgia Institute of Technology*	Dr. Hyesoon Kim *Associate Professor, College of Computing* *Georgia Institute of Technology*
Dr. Santosh Pande *Associate Professor, College of Computing* *Georgia Institute of Technology*	Dr. Richard Vuduc *Associate Professor, College of Computing* *Georgia Institute of Technology*

Date Approved: August 22, 2016

Dedicated to,

my parents and my siblings - my life is nothing without you all.

ACKNOWLEDGMENT

During the course of my PhD, which spanned the last seven years of my life, many people inspired me, contributed technically towards my work, and helped me stay focused in pursuit of my goals. I would like to thank all of them for their contributions that allowed me to achieve my dream.

The most important contribution comes from my PhD advisior, Dr. Sudhakar Yalamanchili, to whom I will remain indebted forever. He has encouraged me to be bold and creative and present new far fetched ideas, listened patiently to every small and crazy thought that I came up with and provided guidance and constructive feedback to each one of them, kept me up-to-date for any new development in my subject allowing my work to remain relevant and state-of-the-art, pushed me to not give up when things are not working as I expected them to, helped me improve my writing, presentation, and communication skills, and even took care of me during my financial and visa problems. He is one of the nicest person that I have ever met and I feel extremely lucky to find such a great mentor in early days of my studies at Georgia Tech. Thank you very much Professor Sudha for enduring me through all these years and helping me shape and reshape my thesis, career, and life goals.

I would also like to thank my dissertation committee members: Dr. Saibal Mukhopadhyay, Dr. Moin Qureshi, Dr. Tushar Krishna, Dr. Hyesoon Kim, Dr. Richard Vuduc, and Dr. Santosh Pande. A few of them have been providing me extremely valuable feedback during the course of my PhD. All of them have given me necessary advices and suggestions after my proposal and dissertation that helped me focus my attention to critical tasks and polish my results. It is the classes taken by these very professors that helped me improve my understanding of the thesis topic allowing me to consider it from different angles. I would also like to thank all my teachers in Georgia Tech and also my teachers during my previous masters and undergraduate studies for teaching me the necessary skills to overcome the challenges of a PhD.

Next, I would like to thank all my labmates / colleagues of Georgia Tech for the support they have provided me in my work. Six seven years is a long time to work, collaborate, discuss ideas with a large number of people. I would like to thank all of them for their constructive comments and thoughts that have shaped up my research and career goals. Special mention needs to be given to Mitchelle and Dhruv from the early days and Si, Eric, and William in the later half of my thesis. Thank you very much for helping and supporting me during crazy never ending deadlines. Special thanks to the developers of the tools, specially manifold and kitfox, that I have used extensively during my research. I would also like to thank my managers, mentors, and colleagues of my internships / part-time jobs that I did during this period specially Luv and his team, Vinod and his team, and Dr. Shoab and his team. In a brief period, you guys have transfered part of your knowledge that not only improved my understanding of the subject matter but helped me a great deal in shaping up my thesis. Thank you very much for listening to my crazy ideas and keep encouraging me to broaden my knowledge base even further. Thanks also to the research collaborators from other universities and departments as well specially Maciej Besta of ETH Zurich for the SlimFly work.

Next, I would also like to thank all the friends that I made during my time in Atlanta. I met some of the most amazing people during my stay and had some unforgettable memories with them. These are the people that keep me positive through difficult times, made me laugh, and allowed me to keep going. The list is very long but special thanks to Hussain, Sajid, and Ali; Muneeb and Abdul Qadir; Saad and Asif; Mohsin and Umer; and Adil, Naveed, and Ahmad. You guys became my mini-family and made my time in Atlanta one of the most memorable days of my life. My utmost gratitude to Hussain, who endured all my frustrations, supported me in every thick and thin from day one, and never allowed me to loose focus.

I am left with thanking my family members. I will start with my siblings: Meiraj, Maryam, Murtaza, Mujtaba, and Maliha. You are the lights in my heart, the people who have always stood by my side, encouraging me, supporting me, picking me up, and never allowing me to loose faith on myself. Talking and chatting with you guys always brings smile to my face. Sorry for not being available in your happiness and sorrows during the last few years because of the crazy amount of work required to finish the dissertation.

And finally my parents, Mumtaz and Sarwat, for all the prayers, love, and sacrifices they have made to make me a doctoral candidate and then patiently waiting for my dissertation to finish. Thanks to my mother, who deep down never wanted me to leave, but still always encouraged me to pursue my goals. My utmost thanks to my father, who sacrificed his own ambitions to allow his kids to pursue long successful careers. I owe everything to you both.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

3D packaging has emerged as a vehicle for scaling system densities and performance due to i) increased inter-tier bandwidth, ii) reduced inter-tier latencies, and iii) ability to integrate dies from different process technologies as a means of customization and hence performance improvement. Moving forward, continued scaling of fine-grained Through Silicon Vias (TSVs) across a 2D cross section can support a large number of memory channels, hence concurrency in the memory system. High concurrency in 3D memory alters the basic relationships between bandwidth, latency, and energy of the memory hierarchy [53, 105]. We re-evaluate these relationships and identify a few key characteristics, optimizing which can significantly improve power and performance of the overall system. These include i) a re-factored memory latency path, in which the difference between the cache and the DRAM latency is reduced, and the network latency becomes a critical part of the overall memory-access latency, ii) an increase in the importance of concurrency management using address space translations to effectively balance various power and performance trade-offs, and iii) a wider thermal operating range of DRAM that significantly impacts its refresh rate and hence, performance. We exploit these observations by proposing architectural modifications and optimizations at various levels of the memory hierarchy with a goal of improving performance and energy efficiency of the overall system.

We first identify the need for fine-granularity and highly-parallelism in 3D memories and characterize its impact on latency in various components of the cache and memory subsystem. We further establish the role of hardware address translations at various levels of the memory hierarchy in regulating locality vs parallelism trade-off in concurrent memory channels and maximize performance [29]. We propose to re-organize the 3D memory hierarchy in order to reduce network traffic and latency. This re-organization is achieved by

coupling the addressing scheme of a distributed last-level cache (LLC) with that of a distributed high-channel-count main memory. To reduce the network latency further, we design a single-cycle *centralized-buffer router* (CBR) [33] that supports high-radix networks with small dependence of buffer area on radix. We extend it to support adaptivity [31] and multiple virtual channels [4]. Furthermore, we evaluate its power and performance advantages for both regular and irregular topologies. Lastly, we analyze the temperature problem of 3D memories and characterize its impact on system performance and reliability. We take two distinct approaches. First, we explore how new technologies, such as microfluidics cooling, can help alleviate temperature and reliability concerns. Second, we study how adaptations to the current micro-architecture can expand the operating range of 3D memories. Specifically, we propose variable-rate per-bank refresh management for 3D stacked memories that exploits the variability in 3D DRAM temperature to reduce refresh power and allow the DRAM to operate at much higher temperatures tolerating hotspots in the memory subsystem.

CHAPTER 1
INTRODUCTION

A fundamental limitation of performance scaling in chip multi-processors (CMPs) is the decreasing memory bandwidth per core, generally referred to as the *memory-wall problem* [79]. The problem has worsened over the years as the number of cores per chip doubled every two years while the external pin bandwidth has not increased at the same rate. It is further exacerbated by the increasing demand of power delivery pins reducing the availability of data pins. Three-dimensional (3D) packaging [119] and silicon interposer technology [89] have emerged as promising new solutions to overcome the problem; mainly because of their i) increased inter-tier bandwidth, ii) reduced inter-tier distance, and iii) the ability to integrate dies from different process technologies as a means of customization. Higher bandwidth is achieved by a large number of fine-grained Through Silicon Vias (TSVs), or thin metallic wires of the interposer, which, coupled with DRAM RAS / CAS trends and the need to distribute TSVs evenly across the dies, points towards an increase in the number of channels, thus concurrency in the memory systems [30]. Smaller inter-tier distance has brought large amounts of data close to compute, which potentially can reduce the ever-increasing cost of data movement and has renewed the interest in concepts like near-data processing (NDP) [71], processing-in-memory (PIM) [121], and memory-centric computing [41]. A common theme with all these developments is the rise of fine-grained memory-level parallelism on-chip opening up interesting new challenges and opportunities. In this thesis, we analyze an exemplar 3D system, identify its major bottlenecks, and propose architectural modifications to remove these limitations with the goal of improving performance and energy efficiency of 3D memories.

A 3D system alters the basic relationships (i.e. design points) between bandwidth, latency, and power of the memory hierarchy [53, 105]. We re-evaluate these relationships and propose architectural modifications and optimizations that adapts to the new challenges.

Specifically, we identify three key characteristics which will be exploited in this thesis.

1. A 3D system consists of highly parallel memories with large number of banks and channels, effective utilization of which trades off locality, parallelism, and energy. These trade-offs can be managed by various types of address translations which can be used to reshape the memory traffic and optimize power and performance of the overall system.

2. Reduced 3D DRAM delay leads to refactoring of the memory latency path which increases pressure on the interconnection network between the memory and the cache hierarchy. Specifically, network latency becomes comparable to DRAM latency.

3. Lower heat removal capability and higher DRAM density of 3D stacks increases their temperature and requires larger number of rows to be refreshed at significantly higher rates. Increased refresh time decreases memory bandwidth availability and higher temperature puts a limit on core operating frequency; both limiting continued performance scaling.

Exploiting these observations requires architectural modifications and optimizations at various levels of the memory hierarchy which is the topic of this thesis; and stated as follows, *”High concurrency in 3D stacked memory systems have severely impacted locality, bandwidth, and energy trade-offs, management of which is critical to fully exploit the potential of 3D memory systems”*.

Following is a brief summary of the major work accomplished in this thesis and is categorized into three distinct parts.

1.1 Analysis and Optimization of an Exemplar 3D System

The first major contribution of this thesis involves analyzing an exemplar 3D system (described in chapter 3), identifying its major bottlenecks, and proposing architectural modifications to address these bottlenecks. The goal is to understand how to carefully manage the parallelism in 3D memories and remove performance inefficiencies by proposing address translations and memory hierarchy re-organization without adding or modifying the internal hardware structures of the core and the memory subsystem. The key insights gained by the analysis of the exemplar 3D system are as follows.

- Assuming a fixed wiring density between the processor and the memory, a 3D system with large number of narrow memory channels has superior performance over a 3D system with a small number of wide memory channels. We use a channel count of 16 for our baseline configuration.

- A large number of channels and banks results in high parallelism, which can be effectively managed by simple address translation schemes at various levels of the memory hierarchy to regulate the power and performance trade-offs of the overall system. A fine-grained interleaving between channels and banks exploits parallelism increasing performance, but destroys any locality in the memory access stream. It also increases the load on the network and increases power of the overall memory subsystem. A coarse-grained interleaving scheme on the other hand does not fully utilize the high bandwidth of 3D memories and creates hotspots in the network.

- High parallelism in 3D memories reduces the load on individual DRAM banks and channels, which results in decreased queuing delay and smaller latency in the DRAM itself; thus putting more pressure on the interconnection network connecting these channels.

- Reduced DRAM latency and high network traffic increases the criticality of network latency making it comparable to that of DRAM latency, an observation that does not hold true in conventional 2D systems. This makes 2D network bandwidth a limiting factor for full exploitation of 3D DRAM bandwidth.

- Address management schemes at various levels of the memory hierarchy can nullify each others benefits. Their careful co-ordination is critical to optimize the power and performance of the overall system.

Taking these observations into consideration we reorganize the memory hierarchy into a banked memory-side cache organization that reduces the network traffic. The re-organization is made possible by the co-ordination of address mapping at the LLC and the MC level. To reduce the network traffic further we implemented locality based OS page allocation strategies that tries to keep the data close to the requesting cores as much as possible and analyze their impact on the overall system performance. We also show the importance of traffic distribution to all the channels in order to utilize maximum bandwidth without violating the conventional principles of spatial locality and DRAM hit rate; and use customized address translation schemes to provide this distribution.

1.2 The 2D Network

The first part of the thesis served to identify i) the importance of the 2D network bandwidth in exploiting 3D bandwidth, and ii) its increasingly dominant role in the overall memory access latency. The thesis focuses on addressing this challenge via the design of a family of interconnection networks that enable effective tradeoffs between area (buffer space), power and bandwidth. The second contribution of this thesis proposes a router micro-architecture suitable for high-radix on-chip networks that reduces network latency and power while maximizing throughput. The goal is to minimize network latency and maximize 2D bandwidth utilization in order to achieve full potentials of high 3D bandwidth. Network latency can be reduced by increasing the router radix and reducing the average hop count of the memory accesses. Higher radix requires large number of links which are also available in abundance on-chip. However, the problem in using large number of links is that each link is associated with multiple buffers, with each one of them taking significant amount of area and power. Furthermore, the minimum depth of each of these buffers has to be equal to the credit round-trip latency (to avoid bubbles between successive flits and maximize link utilization), which also grows with radix and link length. Thus, increasing router radix to reduce latency amplifies buffer needs and exacerbates energy inefficiency.

In this thesis, we propose the use of centralized buffers (CB) in the routers to decouple the required buffer space from its radix and link length. We further present modifications to the baseline centralized buffer router improving its performance and energy and making it suitable for different network configurations. The key contributions in this domain are the following.

- We propose centralized buffer router (CBR) - a router micro-architecture based on the use of centralized buffers (CB) with elastic buffered (EB) links [59] that supports high-radix networks. At low loads, the CB is power gated, bypassed, and optimized to produce single cycle operation. At high loads, flits are streamed through the buffered path taking three cycles. A novel extension to bubble flow control enables routing deadlock and message-dependent deadlock to be avoided with the same mechanism having constant buffer size per router independent of the number of message types.

- Flow control of the centralized buffer routers works at a granularity of individual packets in the central buffers. This increases the minimum buffering space required by the CBs. We reduce the buffering requirement of the central buffers by introducing *Bubble Sharing*, a flow control technique that extends the worm-bubble flow-control [16] scheme to centralized buffer routers, reducing its buffer space.

- We also propose *Adaptive Bubble Sharing* that enables adaptive routing with bubble-sharing flow control for wormhole switched networks. This reduces latency of the centralized buffer routers further, specially in the case of high-radix networks with large number of adaptive paths.

- Centralized buffer routers uses elastic buffer (EB) links with variations of bubble-flow control for deadlock avoidance [32]. However, bubble-flow control works only for regular topologies, such as rings and torus. For complex irregular topologies, we extend the centralized buffer design to support multiple VCs. We used recently proposed ElastiStore [84] technique to allow virtual channel with CBRs.
- To explore VC-based CBRs with high-radix irregular topologies, we used the VC-enabled CBR design with Slim Fly, a recently proposed high-radix topology by Besta and Hoefler [5], for an optimum diameter-2 on-chip network with the least number of ports in the router. The topology is fairly irregular and requires two VCs for deadlock freedom. Hence CBRs with multi-VC support are used.

We evaluated the centralized buffer router design and its modifications with various regular and irregular, and low- and high-radix networks and compared its results with various state-of-the-art routers. The major conclusions of these evaluations are as follows. i) A small central buffer in an EB-like design (CBR) can avoid deadlock and improves throughput without the need of having separate virtual channels and physical networks. The size of this buffer is small which can be further reduced by using bubble-sharing flow control, thus the area and power requirement reaches that of EB with a single physical network. The low-load latency is equal to buffer-less routers and the high-load latency is higher than state-of-the-art buffered routers. ii) CBRs, in general, are superior with high-radix topologies because of the slow rate of increase of their buffer space with that of their radix. They perform equally well with low-radix topologies. However, adaptive bubble-sharing CBRs do not perform well with low-radix topologies because of fewer adaptive paths in them. In such a case, deterministic bubble sharing routers performs the best with very small buffering requirements. iii) CBRs that uses bubble-flow control for deadlock freedom are applicable only for regular topologies, such as rings and torus. For irregular topologies, multi-VC CBRs that does not use bubble-flow control are required which provides the best energy-delay product and throughput per unit power for large networks.

1.3 Thermal Analysis and Optimizations of 3D Memories

The last major contribution of this thesis involves dealing with the temperature problem of 3D memories. 3D stacked DRAMs with their lower heat removal capability, higher DRAM density, and higher heat flux operate at much higher temperature as compared to conventional 2D memories. Thus, their usage has been limited to memory-intensive applications running at low frequencies which inherently have lower performance. Further improvement in performance scaling by increasing core complexity or frequency cannot be achieved as it is limited by thermal constraints of the stack. Increased temperature also has a negative impact on lifetime reliability of the memory system.

To this end, we analyze the thermal profile of an exemplar 3D system and identify that thermal constraints are a limiting factor for full utilization of 3D bandwidth. We also analyze and compare the reliability-performance tradeoffs between multicore systems with 3D and 2.5D stacked memory identifying correlations between frequency, performance, temperature, and reliability; and uses it to quantify the reliability-performance tradeoff. Our analysis indicates that 3D-stacked DRAM provides better performance for most applications, but exhibits poorer lifetime due to higher operating temperature. Furthermore, it also indicates that compute-intensive applications have better performance-reliability trade-offs with 2.5D designs, even at higher frequencies, whereas memory-intensive applications favor 3D stacked systems operating at lower frequencies.

We broadly looked at two different solutions to address the thermal challenges of 3D memories, i) how new cooling technologies, such as microfluidics cooling, can help alleviate temperature and reliability concerns and pushes its envelope to both compute- and memory-intensive applications operating at much higher frequencies, thus utilizing the full potential of high 3D bandwidth, ii) how adaptations to the current memory architecture and algorithms can expand the operating range of 3D memories improving performance and power of the overall system. Specifically, we make a case for variable-rate per-bank refresh management for 3D stacked memories; motivated as follows. Higher temperature of 3D stacks requires large number of rows to be refreshed at higher rates increasing power. Further, different DRAM channels and dies operate at different temperatures due to a high variance in the heat generated by the processor die and a high variance in the heat removal capability of the conventional heat sinks. Exploiting this variation in temperature by allowing different channels and banks to be refreshed at different rates based on their operating temperature not only reduces refresh power and performance penalty but also tolerates thermal hotspots in memory, thus allowing the overall system to operate at much higher frequencies improving performance.

The rest of the dissertation is organized as follows. Chapter 2 gives a brief background of various problems along with the related work in that domain. Chapter 3 analyzes an exemplar 3D system, identifies its performance bottlenecks, and proposes an optimized memory system organization that uses address co-ordination between various levels of the memory hierarchy to reduce the traffic in the network. The next three chapters describes our centralized buffer router (CBR) architecture. Chapter 4 describes the baseline router micro-architecture along with latency and area reduction optimizations. Chapter 5 reduces the buffering space of CBR by proposing bubble-sharing scheme that extends worm-bubble-flow-control to flit level in the CBs. It also proposes adaptive bubble-sharing with CBRs. Chapter 6 extends CBR for multiple VCs and uses it with a recently proposed high-radix irregular topology to reduce the buffering and power requirements of the system. Chapter 7 analyzes the thermal problems of 3D memory and quantifies its reliability concerns along with describing various advantages of microfluidics cooling. Chapter 8 motivates our variable-rate per-bank refresh management scheme and presents its power and performance advantages with chapter 9 concluding the dissertation.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Conventional Memory Systems

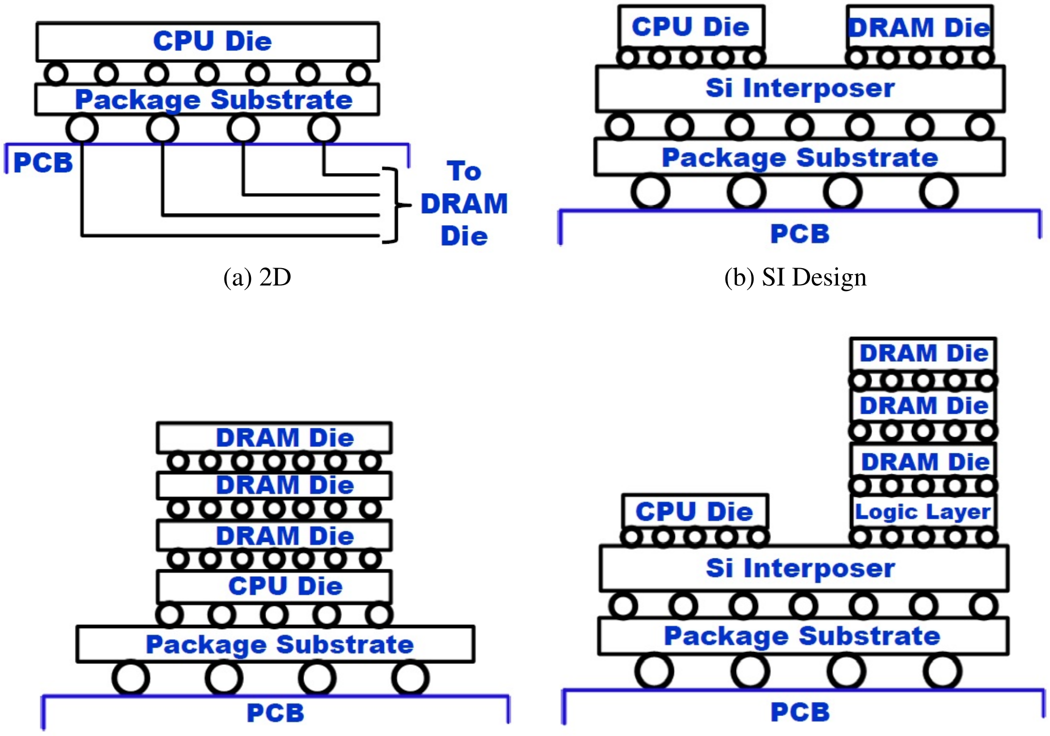
DRAMs conventionally have evolved to consist of multiple ranks (collection of devices operating in unison) that combine to form a DRAM channel [36]. Each rank has many banks which allow multiple outstanding requests to be serviced in parallel by the memory controller (MC). An MC takes incoming requests from the last level cache and issues DRAM specific commands to access the data present in the arrays. The corresponding response is then sent back on completion. However, the number of channels and thus banks are limited in a conventional 2D system resulting in application level interference in the MC. Conventional 2D systems have limited number of memory channels resulting in low DRAM bandwidth and thus application/thread-level interference in the MC queues. Under high load caused by the increasing number of cores and slow DRAM service rates, MCs [65] suffer high queuing delays. To minimize the queuing delays, as well as improve DRAM service rates, several memory scheduling algorithms have been proposed. Their main optimization criteria is to improve either hit-rate or bank-level parallelism (BLP) of the memory requests [78, 65]. Some works have also looked upon optimizing both in tandem in a single-MC case [65], as well as in multiple-MC cases [46]. Yuan et al. [120] uses switch arbitration in the network to increase DRAM hit rate. Das et al. [20] further uses the notion of criticality over hit rate to improve overall system performance. These algorithms improve performance by reordering memory requests going to a bank or across different banks. However, the scope of scheduling is limited, since it is the arrival pattern of requests that govern the delays, and scheduling alone cannot change them.

The arrival pattern of requests can be modified by the use of various address translation mechanisms, which have been explored extensively to spread the data across multiple channels and banks of a DRAM. Zhang et al. [124] uses XOR of bank bits to spread accesses from an application across different banks in a single DRAM. Hassan et al. [29] presents a class of address mapping that distributes traffic to different MCs while preserving the DRAM hit rate. Impulse [122] performs another level of address translation to align consecutive requests and improve hit rate. Micro-pages [99] allocate separate physical DRAM pages to frequently accessed cache lines in order to increase hit rate. Awasthi et al. [3] proposes dynamic page migration from highly congested DRAM channels to less congested DRAM channels that distributes the traffic across different DRAMs. Some other solutions [40] [64] have combined address mapping with memory scheduling to not only distribute the traffic but also reduce the interference caused by various threads on each other. Some groups [35] have also explored using address translations and MC scheduling to reduce DRAM power. All these works improve power and performance of a system by manipulation of memory request in various manners without changing the inherently-inefficient internal structures of the DRAM.

Some researchers have explored changing the internal structure of the DRAM to improve bandwidth or power efficiency. These include approaches like early fully-buffered DIMM [25] and decoupled DIMM [126], which separates the dependence of DRAM bus with its internal storage sub-arrays to improve bandwidth; rank subsetting [1] and mini-rank [125], which reduces the DRAM row size to improve its power efficiency; and SSA [106] and sub-array-level parallelism [47] architectures that propose modifications to the peripherals of DRAM sub-arrays to improve parallelism and reduce energy for each DRAM operation. All these schemes try to improve DRAM efficiency, however, they cannot remove the fundamental challenge of DRAM, that is, the small number of connections between the CPU die and the DRAM chips. In the next section, we provide two promising solutions that address this problem.

2.2 2D vs. 2.5D vs. 3D Stacked Memory

A typical integrated circuit (IC) with a ball grid array (BGA) package consists of a single die placed on top of a package substrate as shown in Fig 1 a). The connections between the die and the substrate, called bumps or more recently micro-bumps, has a pitch around 30-50um while the connections between the package substrate and the printed circuit board (PCB), ball interconnect, is around 400-600um. Although the pitch of micro-bumps can be reduced further, PCB designers are facing numerous challenges lowering the ball pitch down resulting in a very slow increase in the pin count of modern chips. Furthermore, these ball interconnects are connected to long wires in the pcb with non-linear coupling and impedance mismatch effects that keep the speed of these wires low. The combined problem has hindered the rapid increase of pin bandwidth mainly affecting off-chip memory bandwidth and power delivery mechanisms in modern chip multi-processors (CMPs).



(a) 2D

(b) SI Design

(c) 3D

(d) 2.5D

Figure 1: Various packaging options for CPUs and memories

A promising solution to this problem is to introduce another silicon layer, a silicon interposer, between the package substrate and the die and have multiple dies placed together on a single substrate as shown in Fig 1 b). The interconnections between the two dies will have a very small pitch and much lesser distance increasing both the number of connections and their speed. The speed and thus bandwidth is further increased by the fact that the connection between the two dies passes through a much faster silicon layer than through the slow metallic wires of a PCB. Lesser distance also means smaller wires with lower electrical loads and thus smaller drivers to drive these wires, all of which decreases the overall power dissipation. Another advantage of wires in the faster silicon layer is their higher signal integrity leading to removal of sophisticated signal preserving techniques like delay locked loops, on-die termination etc., again improving power efficiency. However, the area and thus cost of the silicon interposer and the package substrate is high.

Another, even better solution to the problem is to stack multiple dies together and make a true 3D package as shown in Fig 1 c). Stacking is enabled by etching holes called through-silicon vias (TSVs) in the bulk silicon portion of the die [7] [24]. TSVs have the advantage that two dies of completely different technologies can be stacked and later bonded using various bonding processes [62]. 3D configuration has several important advantages, such as, reduced distance between the dies, low electrical load on the TSVs, an increase in the number of interfaces between the dies, and a reduction in overall chip area and thus cost. A hybrid of the three technologies can also be used as shown in Figure 1 d). The hybrid configuration has only similar structures like multiple DRAM dies stacked in a 3D configuration with a lower probability of defects due to technology mismatch thus achieving significantly higher yield.

Any organization of various components of a system can be stacked. The most straight forward choice is to stack multiple DRAM dies and place them next to a processor die as a 2.5D structure [118] or put it directly on top of the processor die as a 'true 3D structure' [54]. In this thesis, we will analyze the later case [50] of true 3D stacking, which will simply be referred to as a 3D system. An example of such a design is 3D-Maps [24], which has already been successfully taped out and fabricated. Other common 3D configurations include placing large DRAM- or SRAM-based caches on top of the processor die [105], splitting individual components like processors or caches into multiple layers[116], or even splitting small units like ALU to multiple layers [23]. All these approaches are orthogonal to processor-memory 3D integration and less likely to be adapted in the near future.

2.2.1 Challenges and Trends

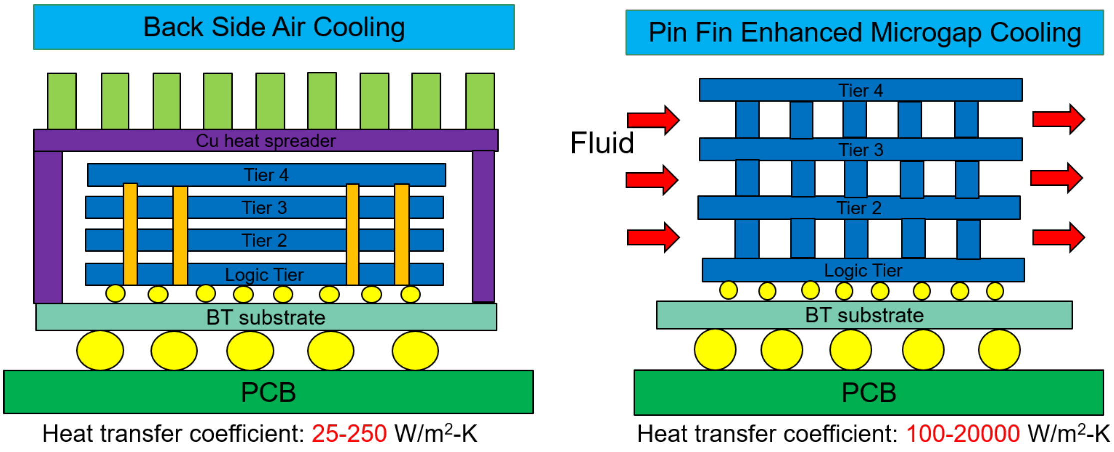
3D processor-memory systems have been gaining significant attention since a decade. Earlier approaches tend to exploit higher bandwidth of 3D, for example, Dong et al [115] have evaluated the use of larger cache lines (equivalent to an OS page) to improve performance. Others [85, 100] have looked towards either restructuring the DRAM array or modifying the interconnect between various DRAM sub-banks in a manner that reduces DRAM access time or energy. However, the adoption of these schemes have been slow mainly because of technology challenges in 1) TSV packaging, 2) thermal management, 3) combined multi-die yield, and 4) DRAM capacity. Recent advancements in packaging [119], microfluidics [110], and silicon-on-interposer [89] technologies indicate rapid progress towards solving these issues leading to the proposal of a few new memory standards [97, 95]. Increasing the capacity of DRAM, however, is still a problem specially with a small number of dies being allowed to stack due to thermal constraints. Increasing the chip size is also not desirable because of lower yield with bigger chips. Consequently, stacked DRAM is mainly been explored as a 2.5D structure [67] with an optional logic die below the DRAM stack. This prompted the architects to come up with interesting new designs, such as, NDC [71] and PIM [121], in which it is used as either a large last-level cache (LLC) [73] or part of a multi-level main memory [88]. Consequently, the focus has been on the management of the DRAM caches or the OS- or hardware-based page swapping between the fast and the slow memory. However, with 4GB of stacked DRAM coming soon [108], the assumption that stacked DRAM will be used only as a cache or page swapping mechanisms are the main performance determinants is less likely to be true. Another problem that arises from large capacity memories is the higher number of refresh operations to be performed, which affects memory bandwidth availability, its power, and the wait time during refresh operations [52]. In this thesis, we revisit the 3D stacked processor-memory system and analyzed it in more detail, identifying its bottlenecks and proposing optimizations to remove them. We focus on the impact of such a system on on-chip network latency, 3D parallelism management and thermal regulation.

2.3 Thermal Concerns in 3D Memory

A major problem with 3D stacks, specially in the case of 'true 3D systems', is their lower heat removal capability, and higher DRAM density and heat flux which increases their operating temperature. Higher temperature limits the operating frequency and requires more frequent DRAM refresh operations (or shorter duty cycles until all DRAM rows are refreshed), both having a negative impact on overall system performance. Furthermore, complex, high power cores are not deemed feasible with 3D stacks as are applications with high compute intensity. Thus, the usage of 3D stacks has been limited to memory-intensive applications running at low frequencies which inherently have low performance. Further improvement in performance scaling by increasing core complexity or frequency cannot be achieved as it is limited by thermal constraints of the stack. We broadly looked at two different solutions to address the thermal challenges of 3D memories, i) how new cooling technologies, such as microfluidics cooling, can help alleviate temperature and reliability concerns and pushes its envelope to both compute- and memory-intensive applications operating at much higher frequencies. ii) how adaptations to the current memory architecture and algorithms can expand the operating range of 3D memories improving performance and power of the overall system. Specifically, we make a case for variable-rate per-bank refresh management for 3D stacked memories.

2.3.1 Various Cooling solutions

Several solutions have been proposed to address 3D stack's heat removal problem. Conventional air cooled heat sinks, as shown in Figure 2a, are placed on top of the stack and has significantly larger form factor than the chip size. Besides the size, the cooling effectiveness or heat transfer coefficient of such systems is low in the range of $25 - 200W/m^2K$ [110]. Other solutions that try to reduce air cooling limitations include thermal-aware floorplanning [18] and thermal TSVs [18] with no electrical characteristics to increase thermal conductivity between the dies and the heat sink. These solutions, although, distribute the heat within or across the dies but does not remove the fundamental problem of high heat flux within the stack. Furthermore, they are not scalable with the increasing number of dies per stack. Another orthogonal solution is to place the high power dissipating logic die closer to the heat sink away from the package substrate but this requires large number of power and ground connections passing through all the memory layers; a solution which is less likely to be practical.



(a) Air Cooling

(b) Microfluidics Cooling

Figure 2: Various cooling solutions for 3D-stacked memories

Liquid cooling using parallel-plate fins called micro-channels [37] has been identified as an alternative to air-cooled heat sinks. Micro-channels are embedded in the inter-layers of a 3D IC. Liquid coolant is pumped through these channels absorbing heat while flowing to the outlet. Micro-channels has been reported to have cooling capability as high as $790W/cm^2$ [37]. A problem in using micro-channels with 3D stacks is that they compete with the TSV area of the dies in the stack. Besides, they require large coolant pumping power. A hybrid of micro-channel and thermal TSVs is proposed to reduce cooling power and still enjoy effective cooling capability [86].

2.3.2 Pin-fin Enhanced Microgap Cooling (Microfluidics)

Although microchannels provide effective cooling solution, recent advancements in fabrication technology has allowed researchers to explore more complex geometries to improve the heat transfer capability of liquid-cooled heat sinks. One promising approach is to have staggered array of micro-sized pin fins distributed across the dies [110] providing localized cooling as shown in Figure 2b. Zhang et al. [128] recently fabricated inter-tier pin-fin enhanced microgap cooling solution, and demonstrated that a staggered pin-fin heat sink with two-die stack can cool at a rate of $100W/cm^2$ with a maximum junction temperature of 47°C. The electrical TSVs are embedded in the pin fins allowing them to not compete with the thermal fins as was the case in micro-channel based heat sink. A compact thermal model to calculate the thermal grids for each of the die was also developed [109]. We have used an extended version of that model in our infrastructure to generate heat maps for different dies in the DRAM stack.

Liquid cooling can be classified as a single-phase or a two-phase solution based on whether the fluid boils and changes its phase while flowing inside the chip or not. We stick to single-phase microfluidics cooling in this dissertation. The flow rate of the coolant determines the cooling capability or thermal resistance of the system which is generally determined by the inlet pressure or pumping power of the system. Zheng et al. [127] shows how thermal resistance decreases with flow rate. The direction of the liquid flow also impacts the effectiveness of cooling. The heat transfer coefficient at the inlet is higher removing more heat than the outlet due to variation in the fluid temperature at the inlet and the outlet. Another important determinant of cooling capability is the pin-fin geometry and height. Ideally, we would like to scatter the pin-fins across the die in a manner that has a uniform heat transfer capability. However, since the TSVs are embedded in the pin-fins, their placement is regulated by the electrical requirements of the TSVs. Zhiyuan et al. [117] explored co-placement of pin-fin arrays based on thermal as well as electrical requirements. Furthermore, higher pin-fin height and larger microgap increases the liquid flow and thus improves cooling. However, electrical TSVs performance are regulated by their aspect ratio, which means thicker pin fins and thus reduced cooling if the pin-fin height is increased further.

2.3.3 Various Refresh Management Schemes

The dynamic nature of DRAM cells means that they lose their charge after a certain amount of time and require periodic restoration through a process known as refresh. A common way to perform refresh of all the cells in the DRAM is to stagger the refresh operations into 8K intervals, generally referred to as distributed refresh [8]. Thus, each refresh operation refreshes multiple number of rows (equivalent to the total number of rows divided by 8K) taking significant amount of time, which is on the rise with larger capacity DRAMs. In conventional DDRx, a refresh operation is performed on all the banks simultaneously making them unavailable during the refresh operations leading to DRAM bandwidth unavailability and reduced performance. Reducing performance penalty of refresh has been a target of numerous publications. Earlier works, such as RAIDR [51], exploits the variation in retention time characteristics of DRAM cells to propose different refresh rates for different rows of the DRAM. The authors in this work assert that only a few weak cells or rows require refresh at a faster rate with most rows allowing a slower rate, thus incurring less performance loss. RAPID [107] uses this idea to partition OS pages into two categories and allocate long-retention-time pages first before allocating the short-retention-time pages, hence minimizing faster-rate refresh operations. Flikker [52] allocates critical data into long-retention-time pages and allows non-critical data to have a few errors.

A problem with all these schemes, however, is that they assume fixed retention time of various DRAM cells determinable statically at the beginning. However, a few cells also show variance in their own retention time, a phenomenon called variable retention time (VRT) [77]. Avatar [72] proposes variable-rate refresh in the presence of VRT cells by periodically redetermining the retention time of cells using memory scrubbing and prevents the intermediate errors using error correction codes. Elastic refresh [98] postpones refresh operations allowing more critical read and write operations to be performed first (more details of this in section 8.4). A similar idea is presented in refresh pausing [66], which pauses the refresh operations allowing critical reads to be serviced immediately before the actual refresh operation can be restored. All these schemes requires refresh mechanism to be controlled from the memory controller which is external to the DRAM. However, JEDEC does not allow fine row-level control of refresh mechanism. Flexible auto-refresh [9] modifies the DRAM control register access protocol to overcome this challenge, thus making other refresh schemes feasible.

More recently, JEDEC has allowed performing refresh at a finer granularity. For example, LPPDDR2 and LPDDR3 standards support per-bank refresh. Kevin et. al [13] shows the advantages of per-bank refresh which allows refreshes to be accessed in parallel with read/write accesses to other banks. Similarly, DDR4 introduces fine-granularity refresh [63]. The idea is to decrease the number of rows to be refreshed with each refresh operation reducing refresh latency but increase the frequency of refresh operations, for example, perform 16K or 32K refresh operations in a 64ms cycle; termed as 2x and 4x modes, respectively. More details of per-bank and fine-grained refresh are given in chapter 8. Partial-array self-refresh (PASR) [21] allows refreshes to be performed for a few banks only ignoring the rest in the self-refresh mode [8], thus saving refresh power.

Another important characteristic of DRAM cells is that its retention time is severely impacted by the operating temperature. Typically, for temperatures below 85°C, DRAMs are refreshed at a period of 64ms and at double the rate for every 10°C rise [60]. However, the rate of 64ms is a conservative value designed for the weak cells to operate at high temperatures. Some recent works have pointed towards decreasing the refresh rate for lower temperatures [82]. A feature called thermal-compensated self-refresh [103] (TCSR) is already provided in modern DDRs which reduces the refresh rate based on the device temperature in self-refresh mode. However, this mechanism is based on internal DRAM refresh counter in self-refresh mode and temperature cannot be regulated from the outside. Our variable-rate per-bank refresh management policy combines per-bank refresh with temperature-aware refresh [28] for 3D memories. It reduces DRAM power and performance penalty and tolerates thermal hotspots in memory, thus allowing the overall system to operate at much higher frequencies improving performance, as will be explained in chapter 8.

2.4 On-Chip Network

Next, we briefly discuss some recent works done in the area of on-chip networks relevant to this dissertation. The state-of-the-practice for on-chip networks is the use of wormhole-switched input-buffered routers that uses multiple virtual channels (VCs). A canonical on-chip router consists of six stages [19]: input buffering (IB), route computation (RC), virtual channel allocation (VCA), switch allocation (SA), switch traversal (ST), and link traversal (LT) organized as a pipeline. Traditionally, the focus has been to combine these stages and reduce the latency within the router. The body flit can skip the RC and VCA stage. The IB and LT stages can be reduced to single cycle specially in the case of single VCs. Similarly, SA and VCA stages can be combined for single VC designs or in the case of speculative routers [68]. Lookahead routing [22] has been used that performs RC in parallel with SA/VC stages of the upstream router and sends that information to the downstream router along with the control path. Thus, the router pipeline has been reduced within the router to only two stages; RC/VCA/SA and ST along with the combined LT/IB stage. Note that LT can be of multiple cycles based on length of the link. Some other schemes like prediction router [56] and [43] speculatively performs SA stage in parallel with IB stage and further reduces the latency within the router to a single cycle only. The overhead of speculation, however, is high.

2.4.1 Topologies

Latency in the network can further be reduced by using high-radix topologies [45, 5] that decreases the number of hops traversed by a packet before reaching its destination. Although, the increase in the number of metal layers and scaling of wire geometries on chip entails greater wiring densities, and thus supports high-radix networks, it also means longer wires, bigger crossbars, and an increase in the router buffer space, thus overall higher power. Hence, traditional on-chip networks have employed low-radix topologies. (tori and meshes). More recent designs aim to achieve small hop counts with low-radix topologies. For example, EVCs [48] enable packets in a mesh-like topology to virtually bypass routers along their paths lowering the energy-delay product. Similarly, MECS [27] uses multi-drop technology to connect routers in one dimension with a single long link. Both requires long wires which may hurt performance. Kilo-NoC [26] combines the advantages of MECS with elastic links. A recently proposed technique called SMART [15] tries to remove the negative impacts of the long wires. It is built on driving links asynchronously and placing repeaters carefully to enable a single-cycle latency in these wires. All these topologies are fundamentally low-radix and thus they deliver limited throughput at high injection rates.

There is a power-performance tradeoff between high- and low-radix topologies. High-radix networks try to minimize their buffer and crossbar space while low-radix networks design for smaller hop count. SlimFly [5] is a recent approach that aim at securing a close-to-optimal tradeoff between radix and diameter, ensuring low cost and high performance for both low and high loads. We proposed on-chip slim-fly along with our central buffering scheme as part of this thesis (explained later) to reduce the buffering space and power of the router while maintaining the network diameter of only 2.

2.4.2 Buffer Area and Power

VC-based routers commonly use edge buffers for deadlock freedom and performance optimizations. The size of these buffers grows with the number of VCs, the length of the wires connecting the routers, and the switch radix. While total storage is optimized for performance, actual buffer occupancies can be very low. Reducing the size of these buffers reduces power and improves energy efficiency, hence explored widely. The main focus has been to either share the buffers among the VCs of a given port, reduce the number of entries per VC [101, 59], or share a central buffer among all the ports (Roshaq [104] and high-throughput shared-buffer NOC [75]) along with a fast bypass path for the low-load common case. Their approach, however, mainly works for packet-based networks. Most of them uses credit-based flow control and has inherent limitations with longer packets. The extreme case includes various versions of buffer-less flow control that removes the input buffers altogether by the use of deflection routing [61]. While networks with such routers can be effective at low loads, packets in these networks can suffer very long latencies and incur high packet reordering costs in data intensive applications with high core counts. Some other techniques include flit-reservation flow control [87] and whole packet forwarding [55] that focuses on the efficient utilization of the input buffers.

2.4.2.1 Elastic Buffer Flow Control

Elastic-buffer networks [58] have recently been proposed which retain the minimal buffering requirement without the use of deflection routing. An elastic buffer (Figure 3) adds a simple control logic to the master-slave latches of a D flip-flop to make them two independent storage locations (2-slot FIFO). EB uses a ready-valid handshake to move a flit forward. An upstream ready signal is sent to indicate a free buffer slot. A downstream valid signal is sent to indicate a valid flit. A flit moves to the next buffer when both are asserted among the two EBs. Pipeline bubbles created with ready-valid handshake are avoided by

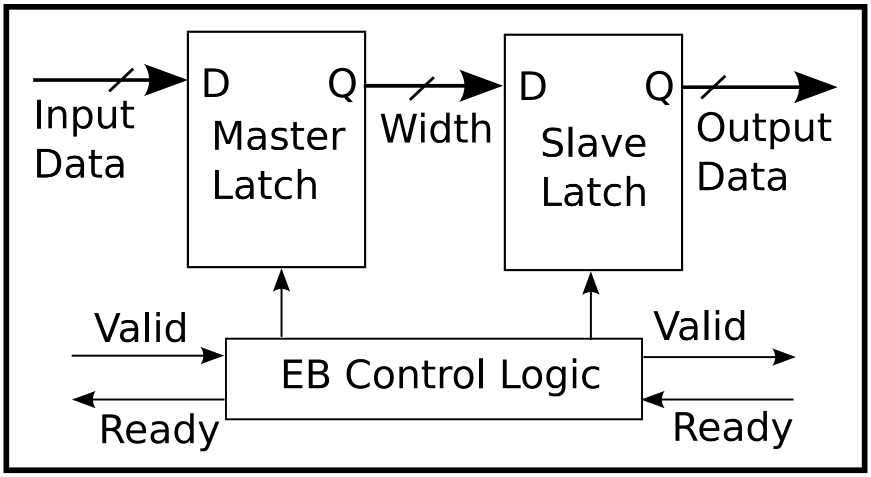


Figure 3: An Elastic Buffer (From [59]). Master slave latches of a flip flop are split along with an associated control logic to make them two independent storage locations.

providing two slots per EB within a single clock cycle delay. Elastic buffers have the property of looking at the next buffer in the pipeline and progress without causing any bubbles in it. Elastic buffered channels are used to provide link level flow control instead of commonly used credit based flow control, which not only requires large input buffers but also creates unnecessary bubbles between the flits. However, pipelined EB links face obstacles in integration with standard performance-optimized router architectures [59]. For example, virtual channels cannot be integrated in the normal manner, and multiple physical channels are recommended for deadlock avoidance. Hybrid EB-VC [59] adds VC buffers to avoid deadlocks in EB. They use a technique similar to on-off flow control for drainage of flits into the VC buffers. Again, on-off flow control requires these buffers to be large enough increasing their area and power. Kilo-Noc [26] uses EB for its MECS topology. They fall back to VC based buffering space in the routers to avoid flits of different virtual channels to cause deadlock. Furthermore, their approach is tailor-made for MECS topology. Elasti-Store (ES) is another recent EB extension that preserves most of the EB advantages while supporting VCs [84]. Our design avoids deadlock in the EB channels without VCs by extending another recently proposed technique known as *Bubble Flow Control* [70] to the central buffers. The scope of our router is much broader as it can work with many different topologies and favors high radix networks with reduced buffering requirements.

2.4.3 Bubble Flow Control and its Variant

Bubble Flow Control (BFC) has been proposed [69] to avoid deadlocks in a packet based ring without the use of VCs. The basic idea is to ensure that at least one packet sized bubble (empty buffer space) is kept in the ring all the time, even after a new packet is injected into the ring. This is ensured locally by allowing injection in any ring only when an empty space of two packets is available at the input port [70]. The problem with localized bubble flow control (LBFC) is that it keeps many empty packets in the ring, incurring long injection delays and reducing throughput. Furthermore, it requires a minimum buffer space of two packets at the input which makes it impractical for on-chip networks. To ensure that only one global empty packet is required by each ring, critical bubble scheme (CBS) [17] was proposed, which marks one packet-sized empty slot in the ring as critical and prohibits injection into the critical bubble. Critical bubbles are moved among the routers both by movement of packets through the ring and by proactive displacement. Both LBFC and CBS, however, only works for packet based networks. A wormhole-based version of CBS called worm-bubble flow control (WBFC) was presented in [16]. The idea is to keep marking flit-sized slots or bubbles in the input buffer as critical before injection, and only inject once enough bubbles are marked to hold a complete packet. In this way, the original critical bubbles of the ring (inserted at initialization) will always be maintained. Some recent work includes dimensional bubble flow control [11] which uses bubble to provide adaptivity in Mesh networks using single VC and [38] which is similar to our packet based flow control extension with edge buffers only. We extended the WBFC scheme to routers with shared central buffers, called *Bubble Sharing*. Both WBFC & Bubble Sharing schemes are further explained in chapter 5.1. CBS is also extended for adaptive routing in bubble coloring scheme (BCS) [113], which maintains one critical bubble in a virtual ring spanning all the routers and uses it as an escape path for adaptivity. More details of the bubble-coloring scheme is given in section 5.2.

CHAPTER 3

ANALYSIS AND OPTIMIZATION OF AN EXEMPLAR 3D SYSTEM

This chapter takes an exemplar 3D memory system, analyzes its performance, and identify the critical importance of lower 2D network latency and efficient parallelism management using address space translations to maximize performance in such a system. We further optimize this baseline system in a manner that reduces network traffic and latency. This chapter only considers address translations and memory hierarchy re-organizations, schemes which can be categorized under careful management of 3D memory parallelism with negligible hardware overhead. Reducing the network latency by optimizing the network itself with novel architectures and topologies will be discussed in the following chapters.

3.1 Summarizing the Experiments and Motivation Behind Them

A 3D system provides a large number of connections between the DRAM and the processor layer which can be used to increase the DRAM bandwidth. However, the new challenge is to efficiently manage this bandwidth, that is, to understand how to best utilize these connections that improves performance and reduces power. To achieve this goal, this chapter evaluates and re-organizes the memory hierarchy incorporating state-of-the-art techniques in 3D stacks.

First, we determine the number of memory channels in a 3D system. We show that having large number of narrow channels is better as compared to a few wide channels. We settled with 16 channels for the exemplar system. It consists of a distributed banked L2 cache as the last-level shared memory with directory-based coherence protocol in which each bank acts as a home directory for part of the global address space. We analyze this system consisting of a large number of channels and LLC banks and point out the high parallelism available in the DRAM, efficient management of which can regulate the power and performance of the overall system. We identify that this massive parallelism reduces the load on individual DRAM banks and channels, decreasing their queuing delay, making the memory-access latency small, and putting more pressure on the interconnection network connecting these channels. This results in the network latency becoming comparable to that of DRAM latency, an observation that does not hold true in conventional 2D systems. Furthermore, we identify that address translation mechanisms at various levels of the memory hierarchy (LLC or DRAM) can severely impact the management of this increased parallelism and used it to regulate locality vs parallelism tradeoffs in the system. We further show that a careful co-ordination between addressing schemes at various levels is beneficial to the overall system performance. Lastly, we also touch upon the importance of traffic distribution to all the channels in order to utilize maximum bandwidth without violating the conventional principles of spatial locality and DRAM hit rate; and use address translation schemes to provide this distribution.

With these observations in mind, we reorganize the memory hierarchy into a banked memory-side cache organization that reduces the network traffic. The re-organization is made possible by the co-ordination of address mapping at the LLC and the MC level. To reduce the network traffic further, we implemented locality based OS page allocation strategies that tries to keep the data close to the requesting cores as much as possible and analyzes their impact on the overall system performance.

3.2 The Exemplar 3D System - An Overview

Our exemplar 3D system consists of a 3D processor-memory architecture, in which multiple DRAM dies are stacked on top of a processor die as shown in Figure 4. The processor die consists of 4x4 tiles, each consisting of two cores + two private L1 caches, a bank of globally shared L2 (divided into 16 banks, one per tile), a router connecting the tiles, and an MC controlling the DRAM dies/layers above. Similar to the HMC organization [67], each of the DRAM layers is divided into 4x4 blocks. One block from each layer combines with the respective block from each of the other layers, creating a DRAM vault. Each vault acts as a multi-layered DRAM with its own memory controller. Hence, there are 16 vaults controlled by 16 MCs in a four-to-eight layered cube. Thus, the basic 3D system consists of 16 DRAM channels (each vault act as a separate DRAM channel), each consisting of four-to-eight ranks (each sub-layer in a vault can be treated as a rank since all share a common bus), and two banks (each sub-layer is divided into two banks that operates in parallel as is the case with conventional DRAMs).

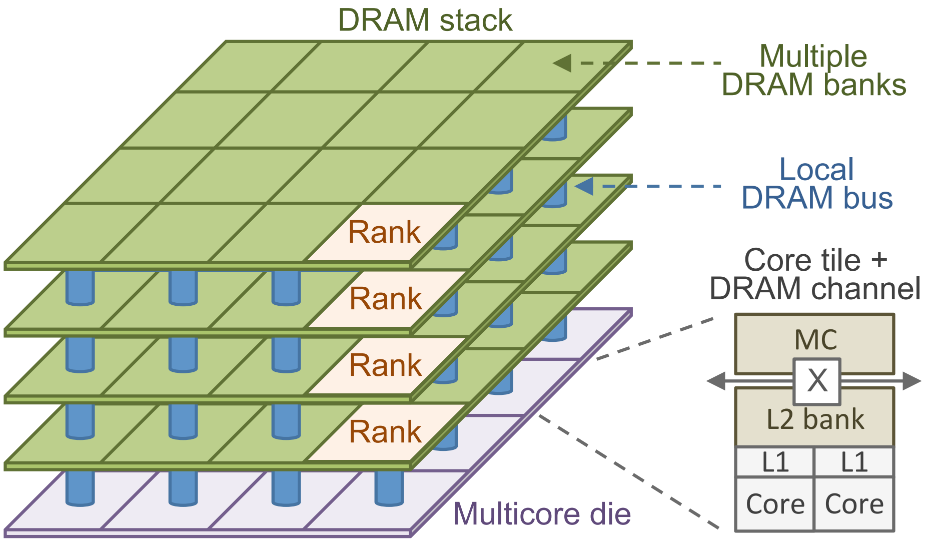


Figure 4: Full system model

3.3 Simulation Environment

We have used the Manifold multicore simulation infrastructure [112] as our simulation environment. Our system simulator is organized as follows. The front-end is a multi-core emulator called Qsim [42] that boots a Linux kernel and executes multi-threaded applications from SPLASH and PARSEC benchmark suites, generating x86 based instruction stream for each thread. These instructions are fed into a multi-core processor timing model. We ran 32 core simulations with one thread per core. The cores are fast forwarded until all of them start running. As explained earlier, two cores are concentrated in one tile. Loads and stores are sent to the two-level cache hierarchy. We used the mcp [6] model for cache simulations that uses distributed directory based MESI coherence protocol as explained in 3.6.1. It can process requests in parallel as long as they belong to different cache lines. Requests to the same line are stored in a stall buffer before that line's coherence state machine is being updated. All requests whether a hit or a miss are first stored in an MSHR that limits the number of outstanding requests per cache. Iris [112] is used as the network timing model that models a two-stage pipeline router architecture with flit level flow control as explained in [68]. Each router connects to two network Interfaces (NIs), one each on the cache and the MC side. The NI converts flits to/from cache level requests in its separate injection and ejection queues. The memory model is constructed using the open source DRAMSim2 memory simulator [81]. 16 vaults correspond to 16 instances of DRAMSim2. A single DRAM vault consists of four ranks (equivalent to DRAM layers) and two banks per layer of 32MB each. Thus the total DRAM capacity of the system is 4GB (each vault = 256MB, each sub-layer per vault = 64MB). TSV latency across different layers is kept constant, which is equivalent to the DRAM bus speed. The bus speed is modeled as twice the DRAM speed. Configuration parameters for various system elements are shown in Table 1. DRAM timing parameters used are given in Table 2.

Table 1: System configuration

Components	Various Parameters & Values
Processor	Out-of-order, 6 stage pipeline, 2GHz, 2-wide issue/commit, 64-entry ROB, LSQ
L1 cache per core (8KB)	32 sets, 4-way, 64B lines, 8 MSHRs, LRU replacement, 2-cyc hit, 5-cyc lookup
L2 cache per tile (256KB)	256 sets, 16-way, 64B lines, 32 MSHRs, LRU replacement, 10-cyc hit, 20-cyc lookup
Network	4x4 torus, request reply, flit-size - 128 bits Pkt-size - 3 flits w/o data, 6 filts with data, baseline x-y routing (2VCs per virtual net.)
Router	6-port, 5 flits IB, 4VCs/port round robin SA, FCFS VCA
Memory controller	rank and bank round robin, close page, Addr-map - chan:row:col:bank:rank
DRAM config. per vault	64M/die/MC, 1-channel, 4-rank, 2-banks, 8KB row, and 64 bit bus @ 1333MHz

Table 2: DRAM timing parameters [97]

Parameter	**Value - cycles (ns)**
t_{CLK}: Clock cycle time	1 (1.5 ns)
t_{RP}: Row precharge time	9 (13.5 ns)
t_{RCD}: RAS to CAS delay	9 (13.5 ns)
t_{RC}: Row cycle time	33 (49.5 ns)
t_{RAS}: Row active time	24 (36 ns)
t_{CAS}: Column access latency	9 (13.5 ns)
t_{WR}: Write recovery time	9 (13.5 ns)
t_{WTR}: Write to read latency	1 (1.5 ns)
t_{RRD}: Row to row active delay	4 (6 ns)
t_{CCD}: Column to column delay	4 (6 ns)
t_{RTP}: Read to precharge delay	5 (7.5 ns)
t_{RFC}: Refresh period	60 (90 ns)
BL: Burst length	8
Refresh Count	8192

The processor, the cache, and the network execute at 2GHz and the MC (DRAMSim2) executes at 1GHz. We track the Global Instructions Per Cycle (IPC) - the total number of instructions executed across all threads divided by the total number of cycles, which in these experiments is set to be 100M (~1200-2500 million instructions). Next, we will discuss the impact of various characteristics of the exemplar system on the overall system performance.

3.4 Impact of the Number of Memory Channels

A basic consideration in arranging a 3D memory system is to determine the number of memory channels in the system, e.g., if we assume 1024 data I/Os, should they be arranged as one 1024-bit wide channel or sixteen 32-bit wide memory channels distributed evenly across the die. Note that the bandwidth of the overall system remains the same in both the cases. The following two subsections briefly explain why having multiple channels is a better choice for future memory systems.

3.4.1 Larger BL and Smaller Transaction Wastes Memory Bandwidth - All Hits Case

The internal speed of DRAM technology is not scaling fast enough, therefore, keeping the fundamental latencies of activate, precharge, and CAS operations constant around 13-18ns across various generations. Similarly, the row cycle time still hovers around 50-60ns. The DDR bus speeds, however, maintained an upward trend from early 400MHz DDR2, to 2133MHz, and even 3200MHz LPDDR4. Such bus scaling has been maintained by over-fetching the data internally and then delivering it to the DDR bus in a burst mode, which has increased the minimum burst length (BL) parameter from two-to-four in early DRAMs, to eight-to-sixteen in modern DDRs. Assuming fixed bus widths, larger burst length results in higher minimum transaction size (e.g., the minimum transaction size of x32 LPDDR4 with burst length of 16 is 64B). On the other hand, cache line sizes are not increasing at a fast rate, and with the advent of on-chip accelerators, the size of transactions being requested from the memory is decreasing as well. As a result, part of the fetched data will be wasted, e.g., for a 32-byte transaction with the above mentioned LPDDR4 technology (minimum transaction size of 64B), half of the memory bandwidth will be lost even with all page hits. This is shown in an example in Figure 5, where a request stream of 32B with a stride of 64B is accessing an LPDDR4 with a minimum transaction size of 64B. Although, the DRAM is observing 100% hit rate, half of the data being fetched at each transaction is thrown away without being utilized reducing the memory bandwidth utilization to 50%. In such a case, reducing the bus width to x16 (minimum transaction size of 32B) will remove the bottleneck, allowing the memory to have two separate channels that can operate independently. It should also be noted that the problem will worsen in the case of writes, in which masking the other half of the bytes will require reading the data first. We would also like to note here that this is not a very uncommon case and streams with stride greater than 32 will face this issue.

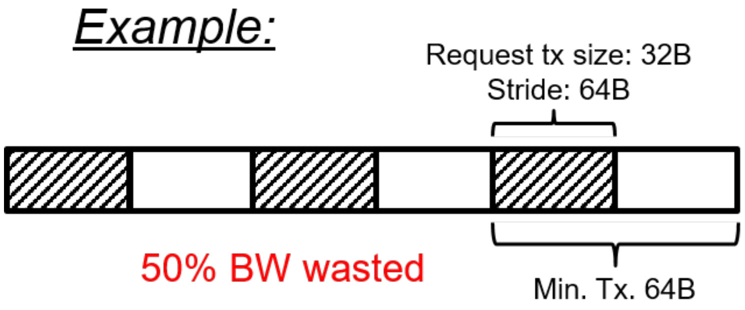


Figure 5: Effect of Number of Channels - All Hits Case

3.4.2 Increased Frequency Reduces DRAM Efficiency - All Misses Case

As mentioned earlier, the DRAM bus frequency has been increasing rapidly with modern DDRs. A problem with high frequency is that a constant row cycle time (t_{ras}) of 50-60ns translates into a larger number of cycles. However, the number of cycles in which the bus is actually transferring data (the data cycles) for a given transaction size remain constant. Thus, in the case of consecutive page misses, in which the next transaction can be sent after a minimum of t_{ras} cycles, only a few of the bus cycles will be transferring data. For example, in Figure 6, the data transfer time with 800MHz DDR is larger than 1.6GHz DDR. However, the time after which a subsequent miss to the same bank can be served is same in both the cases. Thus, the efficiency of DRAM will be lower with higher frequency in high miss rate scenarios. The problem will be worse with large bus widths. For example, in the case of x1024 bus with $t_{ras} = 60ns@2133MHz$, every subsequent miss transaction can be sent only after 128 cycles, and the DRAM transfer rate is 128B per data cycle. Even for a transaction size of 128B, all cycles except the first cycle will be wasted. Hence, the DRAM in the all page miss case has a maximum efficiency of $1/128 < 1\%$ with a parallelism of only one. On the other hand, if we use thirty two x32 buses instead of one x1024 bus (32 data cycles for 128B transactions) and 32 parallel channels, the DRAM efficiency will jump to $32/128 = 25\%$ without wasting any data burst cycles, and allows for a maximum parallelism of 32.

It should be noted that reducing the bus width increases the latency in the bus (one data

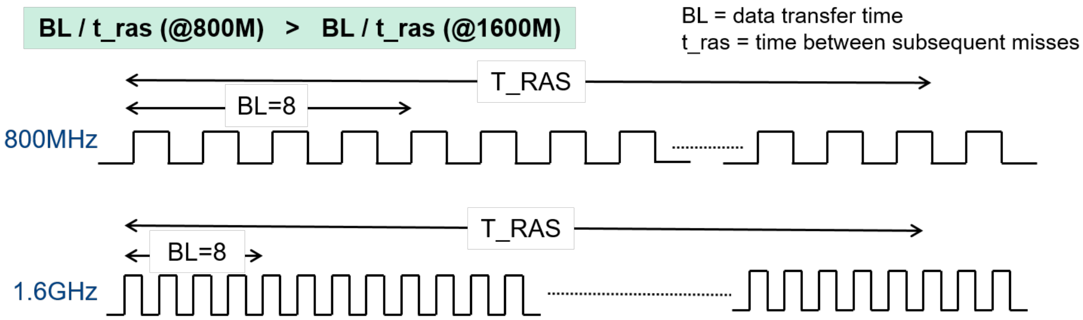

BL / t_ras (@800M) > BL / t_ras (@1600M)
BL = data transfer time
t_ras = time between subsequent misses
T_RAS
BL=8
800MHz
T_RAS
BL=8
1.6GHz

Figure 6: Effect of Number of Channels - All Misses Case

cycle vs. 32 data cycles), which may decrease the system performance. However, the time on the bus is a relatively small portion of the overall DRAM latency, e.g., 32 bus cycles @2133MHz = 15ns of the 100 or more nanoseconds taken by the memory in general. The rest of the time is consumed by activate, precharge, and CAS operations, along with the queuing delays. Hence, increasing the cycles on the bus impacts the overall system performance only by a small amount, even with a large number of DRAM banks sharing the bus. Again, this case can also occur frequently with interference among multiple threads and processes. Not only that, all other cases, which occur between these two extremes, will face similar concerns.

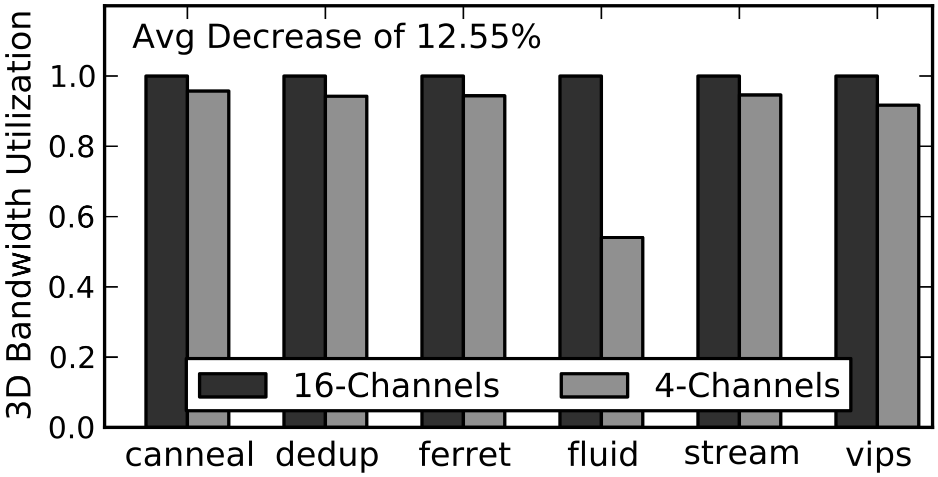


Figure 7: 3D bandwidth utilization with various number of channels

We performed simulations with four and sixteen channels having bus width of 256 and

64 bits, respectively. We increased the number of ranks by 4x in the 4-channel configuration to reduce the impact of the queuing delay. The rest of the system is configured similar to the baseline model discussed in section 3.3. Fig 7 shows the 3D bandwidth utilization of the two configurations. It can be seen that all benchmarks improve their performance by increasing the number of channels. Overall, reducing the number of channels decreases 3D bandwidth utilization on average by 12.5%.

3.4.3 Cost of a Channel

The previous analysis shows that increasing the number of channels is extremely beneficial for performance reasons. However, a problem with increasing the number of channels is the reduction in DRAM density. Conventionally, DRAM is considered a commodity device which maximizes its density to reduce cost. However, increasing the number of channels and banks (aka parallelism) reduces the size of internal DRAM arrays and subarrays, increasing the area taken by decoding, sensing and other peripheral logic attached to it. All channels operate independently with all the peripheral circuitry, including the memory controller schedulers, are replicated with each channel increasing its cost. We haven't explored the area cost of having large number of channels and fixed our basic design to 16 channels, a number inspired by hybrid-memory cube (HMC) 1.0's internal DRAM structure [67].

3.4.4 Current Standards

In this section, we briefly point out how modern DRAM standards are coping up with the above mentioned challenges. LPDDR4 [96] supports both very high speed and a burst length of 16. However, it keeps the bus width small (x16, x32) increasing the number of channels. DDR4 [94], which again has very high speed, introduces the concept of bank groups [2]. The idea is to keep the burst length of eight (4 DRAM cycles) but does not allow another read/write operation to the same bank group for eight DRAM cycles. The other bank groups, on the other hand, can accept read/write command after only four cycles, as is the case with conventional DDRs. (This is achieved by having two separate t_{CCD}. t_{CCD_L} for same bank group and t_{CCD_S} for different bank groups). The problem with this approach, however, is that consecutive hits to the same bank group can only operate at 50% bus utilization. Wide I/O [93] has low speed and thus very large bus width with only one-to-two channels supported. However, Wide I/O 2 [97] has increased the number of channels to four-to-eight allowing the bus width to reduce (with larger prefetch) and increasing the frequency. At this rate, it is very likely that Wide I/On will support even more channels. HBM [95] again supports eight channels and HMC 3.0 internally has migrated to 64 vaults (each vault can be treated as one DRAM channel) with each die partitioned into 64 sub-dies (one for each channel) similar to the exemplar system.

3.5 Impact of High Bandwidth

Next, we discuss the impact of high DRAM bandwidth on various other components of the memory subsystem.

3.5.1 Smaller Cache Sizes and Fewer Levels

We build an analytical model to test the impact of 3D bandwidth on the overall system [76]. The results indicate that smaller 3D DRAM latency increases cache's miss tolerance and achieves the same performance with smaller caches and only two levels of the cache hierarchy as compared to a larger three-level cache hierarchy in a 2D system. This means that more chip area and correspondingly power budget can be devoted to compute than storage further improving performance.

3.5.2 Reduced MC Queuing Delay

As discussed earlier, the fundamental RAS/CAS latencies in conventional DRAMs remained roughly the same over the years, since the time to charge and discharge the DRAM capacitors did not change significantly across technology generations, a trend that is persisting with 3D as well. Only, a very small reduction in RAS/CAS latencies will be seen

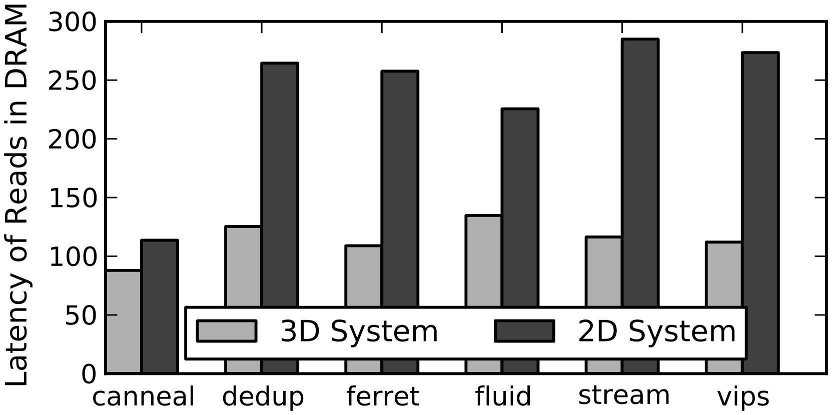


Figure 8: 2D vs. 3D system DRAM latency (cycles)

because of the sharing of some peripheral logic. On the other hand, bandwidth and parallelism will increase, which will lead to reduced memory traffic per channel and thus reduced queuing delays in the MC. Since the queuing delay is a major component of the DRAM latency, the overall round-trip time decreases. Figure 8 plots the latency of reads in DRAM with a 4-channel 2D and a 16-channel 3D system with the same number of cores and caches. Note that the RAS/CAS latencies in both the systems are kept the same. On average, the DRAM latency of the 2D system is 2.1 times higher than that of the 3D system. The main contributors of this increase in latency is the increased memory interference in the MCs (a result of higher memory traffic) and an increased queuing delay.

3.5.3 Comparable Network Latency

Due to reduced queuing delays, network latency in a 3D system becomes comparable to that of the DRAM latency. This behavior is shown in Figure 9, which plots the latency distribution of DRAM-bound read requests for various PARSEC applications. Note that in the figure, $L1-L2$ and $L2-DRAM$ correspond to the latency in the network, and DRAM-only corresponds to the latency in the DRAM. The impact of DRAM latency on the overall system is further reduced by the fact that the ratio of the memory traffic that reaches the DRAM is generally around a quarter to one third of the total number of requests that travel the network (e.g., 25% for vips). The remaining misses are satisfied by remote caches and

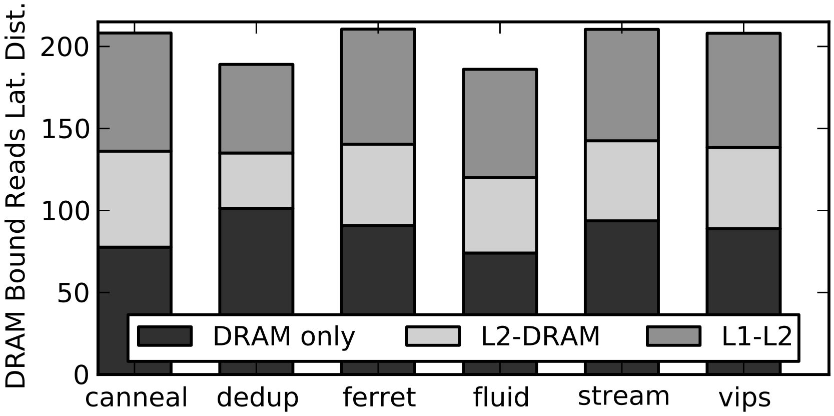


Figure 9: Latency of requests from DRAM (cycles)

related coherency traffic, which only travel in the intra-die links.

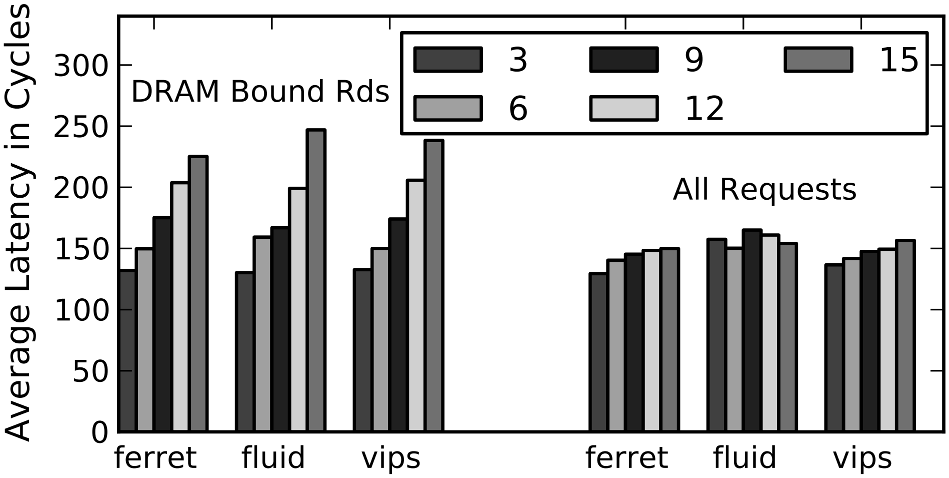


Figure 10: DRAM-bound read latency vs. CAS latency. CAS in DRAM cycles, Request in CPU cycles

To further explore this point, we use a parametric model in which various DRAM timing parameters are normalized relative to the CAS latency. The model is used to illustrate the behavior of DRAM bound reads as a function of CAS latency as shown in Figure 10 (left). The impact on the global IPC is illustrated in Figure 11. From the two figures, it can be concluded that although increasing the latency of DRAM accesses (CAS) increases the overall latency of DRAM-bound read requests, the average latency of all L1 misses (Fig 10 right), and hence the IPC, does not decrease at the same rate. On the other hand, the latency of the 2D network affects all requests (memory bound and coherence included), and therefore contributes more towards the overall latency penalty.

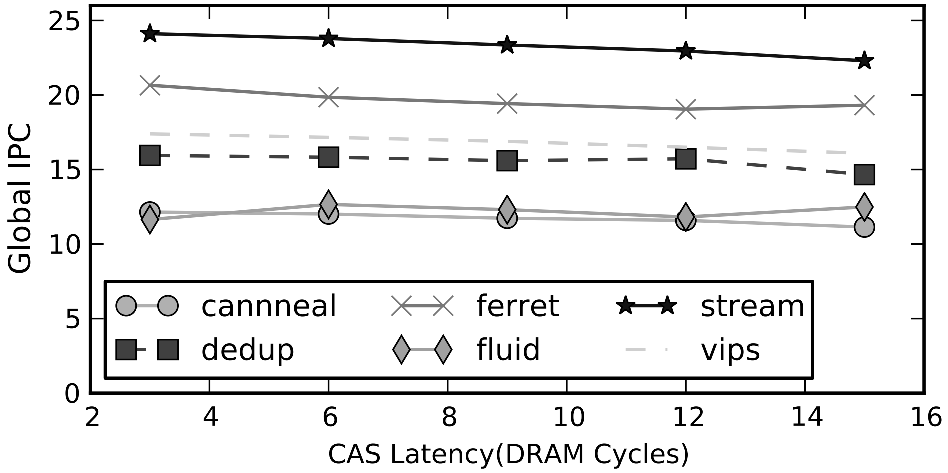


Figure 11: CAS latency vs. IPC

3.6 Impact of Address Translations

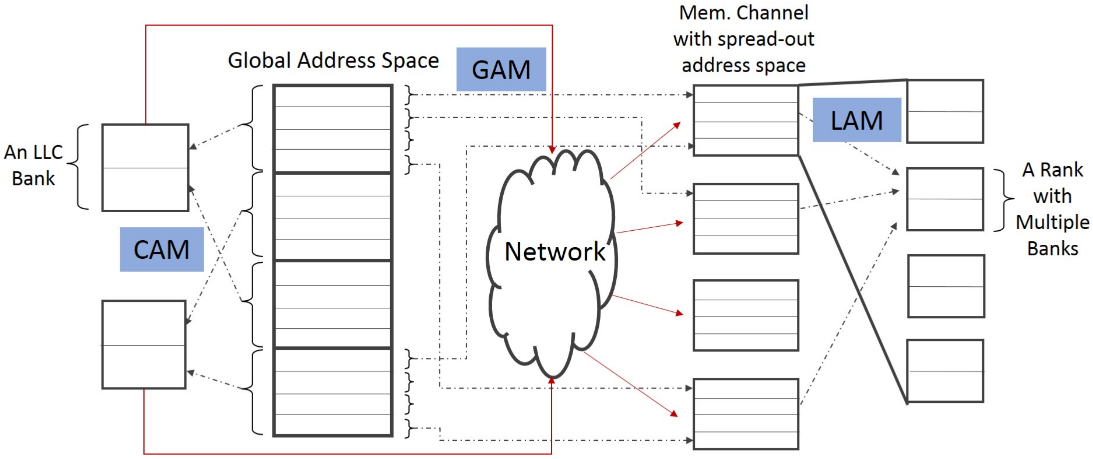


Figure 12: Global address space distribution in cache banks and DRAM channels/banks

The exemplar system has multiple L2 banks distributed across the tiles with directory-based coherence protocol. In such a system, part of the address space is assigned to each L2 bank, and the corresponding bank is treated as a home directory for the assigned addresses [39] [80]. Similarly, in systems with multiple DRAM channels, the address space is distributed among the channels, or in this case, among the DRAM vaults (Figure 12). The address translation mechanisms that determine the home L2 bank and the corresponding DRAM vault for a given physical address (Figure 13) dictate the amount of parallelism and the number of hops in the network. For DRAMs, the interleaving granularity of address mapping functions determines whether the memory traffic is distributed among all memory channels at a particular instant of time (low-order interleaving), or localized to one channel (high-order interleaving); see Figure 13. For L2 banks, the address assignment policy is governed by the principle of locality, such as, the commonly used first touch policy that keeps the data local as much as possible. The design goals at both levels, however, are conflicting, and thus the interaction of these address translations brings interesting behaviors that has not been discussed in the past, which becomes even more important in the case of 3D systems where we have large number of DRAM channels and L2 banks.

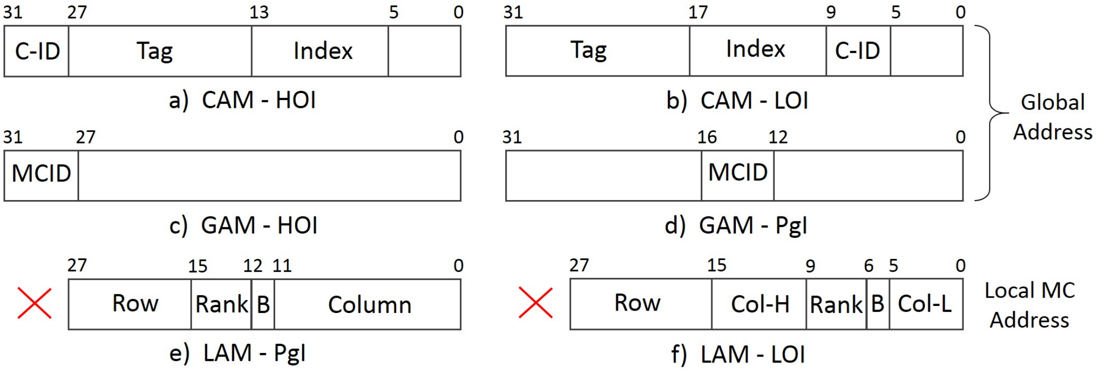


Figure 13: Various address space mappings with CAM, GAM, and LAM

The most straight forward translation, known as high-order interleaving (HOI), maps large blocks of continuous physical addresses to same L2 bank or DRAM vault. On the other hand, low-order interleaving (LOI) distributes cache-line size blocks to different banks while page-interleaving (PgI) distributes the address space at the granularity of OS or DRAM page-sized blocks. HOI preserves any locality present in the application. However, since applications operate in a small address space in a short instant of time, it can place a high load on a particular L2 bank or a DRAM vault during that time instant while all others remain relatively free. LOI, on the other hand, provides maximum parallelism. However, it destroys any locality available in the application. Furthermore, it disperses the network traffic in all directions putting high load on the network. PgI, a commonly used design choice, preserves most of the locality of an application resulting in higher hit rates and shorter DRAM access time. Again, consecutive cache line misses, e.g., in the case of a linear array scan can put higher load on one DRAM vault.

Since all interleaving schemes can happen at both the L2 bank and the MC level, we need to distinguish between whether one is applied at the L2 bank or the MC level. We termed the address translation mechanism at the L2 bank level as CAM (cache address mapping) and the address translations at the MC level as GAM (global address mapping), respectively. Address translations within a DRAM that decides a particular rank and bank are called LAM (local address mapping); see Figure 12. Hassan and Yalamanchili [29] provides more details of various address mapping functions.

The interaction of these address translations at the L2 bank and MC level brings interesting behaviors that has not been discussed in the past. This becomes even more important in the case of 3D systems with large number of DRAM channels and L2 banks. It should be noted here that address translations of both L2 bank and DRAM vault have different design requirements. For example, DRAM does not require multiple parallel requests to same banks and ranks as DRAMs have a long access time which cannot be pipelined. Thus all requests have to be serialized. On the other hand, L2 tag match can easily be pipelined and thus multiple requests to different lines are possible. However, multiple requests to the same line has to stall and reside in some buffer before the first one finishes. This is because they have to wait for the updated coherence state machine of that line to decide what to do next. Next, we show how these address translations can regulate the amount of network and DRAM traffic.

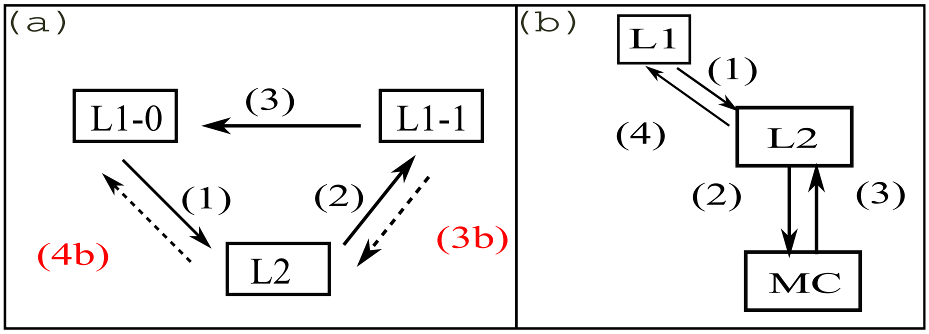


Figure 14: Messages generated to fetch from a) a remote cache and b) the DRAM

3.6.1 CAM and GAM under Multi-level Cache Hierarchy

In a system with private L1 caches, and a distributed banked L2 cache, an L1 cache miss travels through multiple hops in the form of various data- and coherence-related messages in order to get serviced. The important messages among them are indicated in Figure 14. When an L1 encounters a miss, the home L2 bank is consulted for the updated copy (shown in Figure 14 (msg. 1, fig. a)), which can be local or remote based on the address. If the L2 bank experiences a miss, it sends the message to either another L1 that holds the updated copy (msg. 2, fig. a), or to the corresponding MC (msg. 2, fig. b). The MC generally sends the reply back via the L2 (msg. 2/4, fig. b), while the L1 either returns the copy directly to the requesting L1 (msg. 3, fig. a), or sends it to the home L2 first, which in turn sends it back to the L1 (msg. 3b/4b, fig. a). We use the former case of L1-L1 traffic, in which the L2 also requires an acknowledgment to update its state machine (not shown in the figure). The same thing also happens with other requests like invalidations etc. that do not transfer data but update the cache state machines.

The number of hops that each of these messages have to travel is dependent upon the following four factors. i) distance of the home L2 bank with respect to the requesting L1 cache, ii) distance of home L2 bank with respect to the owner of the cache line, iii) distance of the MC with respect to the home L2 bank and iv) distance of the owner with respect to the requesting cache. The first two are dependent on thread assignment to different cores and CAM. The 3rd one is dependent upon the interaction of CAM and GAM. The 4th one is dependent on thread assignment alone. Thus, both CAM and GAM are important to reduce the average hop count of an L1 miss. Note that we have ignored acknowledgment messages in our discussions since they do not lie in the critical path. It should also be noted that GAM which distributes the traffic across different DRAMs is extremely important to reduce latency in the DRAM itself by reducing load in each channel and hence their queuing delays.

The above discussion indicates that each L1 miss results into multiple messages that travel along the network in both horizontal and vertical directions. The hop count of these messages can be reduced by careful co-ordination of CAM and GAM schemes. The result is a modification in the organization of the memory hierarchy explained in the next chapter.

3.7 An Improved 3D Memory System

This section builds on the analysis done in the previous sections and proposes a re-organization of the memory system that reduces the network traffic, made possible by a careful co-ordination of CAM and GAM. We further describe the importance of traffic distribution among different memory channels that maximizes parallelism while trying to keep it to the local banks or MCs to reduce the traffic in the network.

3.7.1 Same Address Mapping and Memory Side Cache

In this section, we propose the use of the same mapping function for both CAM and GAM, which is possible only in the case of 3D systems that has comparable number of L2 banks and memory channels. We show the usefulness of the policy by noting the fact that the use of same function for CAM and GAM keeps the L2-to-MC traffic local, that is, only in the vertical direction. The L1 requests have to already travel horizontally in the 2D network (by 2D network we mean links in the horizontal direction) in order to reach the corresponding L2 bank.

The proposed scheme is described in Figure 15. Figure 15 a) represents a 3D system which consists of different functions for CAM and GAM. An L1 miss first travels to its

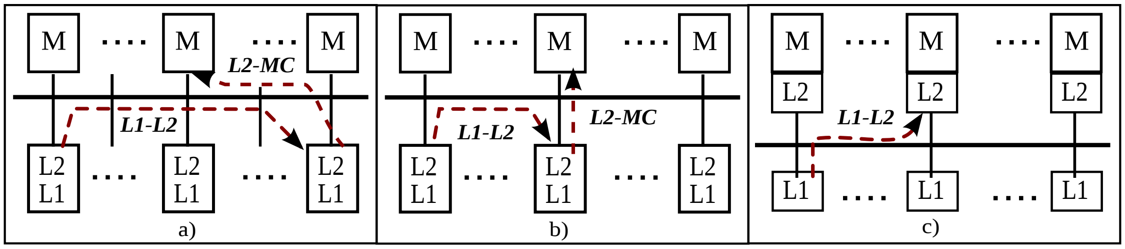


Figure 15: a) The basic exemplar system, b) same CAM and GAM organization, and c) mem-side cache organization

home L2 bank, which is located on a remote tile to determine the current state of the cache line. If the line is not present in the L2 or any of the remote L1s, it is forwarded to the MC, which again is located in a remote DRAM vault, incurring an additional latency in the network for the L2-MC traffic that could have been avoided by using the same function for both CAM and GAM. Figure 15 b) represents the scenario, in which the L2-MC traffic remains local. Furthermore, since there is no remote traffic between the L2 bank and the MC, providing a direct connection between them (as shown in Figure 15 c)) will remove the serialization latency of an L2 miss destined to the corresponding MC through the router (i.e., a miss has to be converted back and forth from/to flits while traveling through the router, and a direct link will remove this breakdown). It also reduces the load on the corresponding router and the NIs. As shown in Figure 9, this latency is 20-25% of the overall memory latency of the DRAM bound reads, and will be removed altogether.

The result is a memory organization, shown in Figure 15 c), in which the L2 is placed closed to the memory rather than the L1, making it similar to the memory-side cache organization. However, the cache is now banked into smaller units with no direct paths among these banks. Two configurations of the organization can be explored, one in which the link between the L1 and the local L2 cache bank is maintained, and the other in which it is removed. Although, removing the link will reduce performance by increasing the latency of requests destined to the local L2 bank, the reduction is not significant. On the other hand, the organization with no L1-L2 link is highly attractive, as it is extremely modular and scalable with each component being designed separately in its own die. Multiple dies can even share a single component by splitting it into more sub-components, one in each die.

3.7.2 DRAM Traffic Distribution - Neighbor Mapping

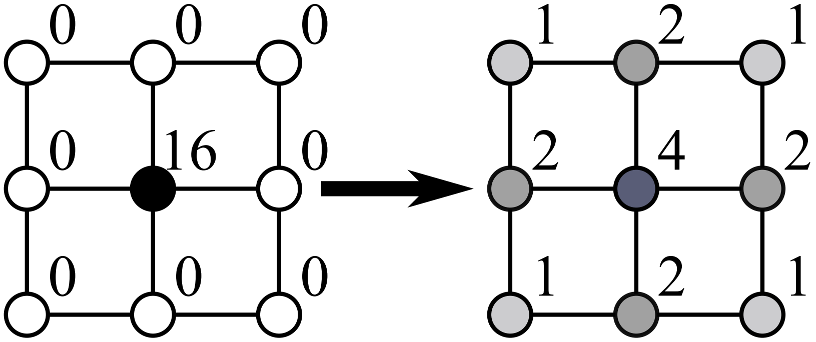


Figure 16: Distribution of pages to neighboring nodes in neighbor mapping scheme

Memory-side cache organization can potentially lead to congestion at a particular DRAM at a particular instant in time, as can be seen with the result of fluidanimate in Figure 18. For such cases, it may be better to distribute the traffic into different MCs, thus not using same mapping function. To explore this point, we defined an address space mapping that distributes pages mapped to an L2 bank or DRAM vault to neighboring banks or vaults. The idea here is to reduce congestion at one MC or L2 bank and increase memory level parallelism by spreading references in a local area that does not increase average hop count significantly. The goal is to better balance parallelism in memory references while keeping DRAM row buffer locality intact. We will refer to this as neighbor-page mapping or simply neighbor mapping. Figure 16 shows how neighbor mapping is performed. Each value in the figure represents the number of pages out of 16 consecutive pages of a particular bank or vault that are mapped to the neighboring banks or vaults. We are not giving details of which particular page is mapped to which immediate neighbor but it is decided based on four bits of the address and is consistent across all banks or vaults, making it a one-to-one function.

3.7.3 Keeping Data Local

From earlier discussions, it is clear that traffic on the network is a major component of the memory latency path. However, our exemplar system distributes the traffic evenly across all the distributed L2 banks and MCs. In this section, we leverage the virtual-to-physical page allocation strategies in order to maximize keeping the data to the local MC and the corresponding L2 bank. This reduces remote L1-L2 accesses, making a large number of them local, and resulting in an overall decrease in the average number of hops. Since, page allocation happens at the OS level, it is completely orthogonal to CAM and GAM. The challenge lies in the placement of the shared data which is accessed by multiple tiles simultaneously and at different program phases.

3.7.3.1 First-Touch Policy

A commonly used scheme that tackles this issue is the first touch policy, where pages are mapped to the banks or MCs whose corresponding cores access it for the first time. This will make all subsequent accesses of the data by that core local; making private only data to remain almost always local. Subsequent accesses for the shared data that happens at the other cores access it as a remote memory.

3.7.3.2 Sharing vs. Replication

We have not explored data replication in multiple L2 banks. Our address space is distributed among all the L2 banks. This can potentially lead to higher remote L2 accesses. Data replication could have been used to reduce these accesses. However, we would like to point out that data replication at multiple L2 banks would have complicated the support for the memory-side cache organization, that is, although the data would have been replicated in the L2 banks, it would be in a single location in DRAM, resulting in remote L2-MC accesses. This would again increase the network traffic and its latency. The choice boils down to reducing remote L1-L2 traffic with a decreased L2 capacity (due to data replication) vs removing the remote L2-MC traffic completely with an increased cache capacity; an analysis to be performed in future.

Similarly, we have not explored the option of migrating data from one MC to another. This can potentially reduce the remote L2 accesses if the program shifts from accessing data from one tile to another in long phases. This is orthogonal to our design and can be explored in future.

3.8 Performance with the Improved Organization

A good metric to characterize applications is by calculating their average hops per kilo instruction (HPKI). HPKI is a product of commonly used misses/per kilo instruction (MPKI) and the total hop length of messages created by each miss. We have defined hop length as the number of hops taken by a request to return. This does not include the acknowledgment messages that are not in the critical path. The network interface is also considered one hop. HPKI captures both the bandwidth requirement from the lower level memory subsystem and the load on the 2D network. The hop length parameter characterizes application locality. It also captures the communication requirements of the coherence protocol. It depends upon address space mapping (at both software and hardware level), network topology and OS thread-core allocation. The MPKI parameter computes the memory vs. compute intensive characteristic of the application and depends upon the L1 & L2 cache sizes. We have used HPKI as it directly correlates with the network bandwidth requirements of the applications under fixed link widths.

3.8.1 Address Mapping - Results

Figure 17 shows the Global IPC values for PARSEC applications with various address mapping functions. In the last three cases, both the shared L2 banks and the DRAM vaults use the same address mapping, which are either high-order interleaving (HOI), low-order interleaving (LOI), or page interleaving (PgI). The first two bars correspond to the use of a different mapping functions for both CAM and GAM.

We can see that the configurations that have different CAM and GAM do not perform

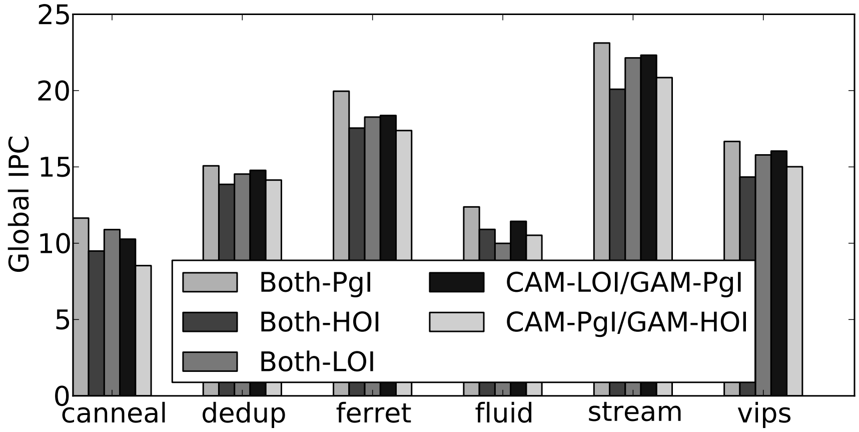


Figure 17: IPC with vaious address mapping functions

well. The average increase of Both-PgI from CAM-LOI/GAM-PgI and CAM-PgI/GAM-HOI is 6% and 14%, respectively. This can be explained with Table 3, which illustrates latency values for different components in vips for all the five mapping functions. The average number of hops (Row 1 of Table 3 have increased significantly in the first two cases, which increases the overall network traffic, and hence reduces performance. Among the three cases with same address mapping for the L2 banks and the DRAM vaults, PgI performs the best. Table 3 illustrates the loss in performance for HOI and LOI, which is 14% and 8%, respectively, in comparison with Both-PgI. HOI will direct most of its requests in a particular time frame to a particular L2 cache and DRAM vault. This will put that DRAM under high load resulting in an increased queuing delay or DRAM latency (last row in the table). Furthermore, it creates network hot spots around that particular node causing an increase in average latency in the network. LOI destroys any locality present in the memory reference stream, thus increases latency within the DRAM. It also disperses the traffic across the network putting higher load on it that results in an increase in the network latency. PgI, which balances locality, average hops, and the DRAM load distribution, performs the best. The results indicate that apart from using same address mapping for both CAM and GAM, distributing the traffic among various DRAM channels without increasing the average number of hops per request is also desirable. All subsequent results use PgI for both GAM and CAM unless otherwise stated.

Table 3: Latency Distribution with different Address Mappings for vips

Latencies (cycles)	**CAM-PgI GAM-HOI**	**CAM-LOI GAM-PgI**	**Both-HOI**	**Both-LOI**	**Both-PgI**
Avg. Hops	11.65	11.94	10.61	10.70	10.68
All Req.	188.25	153.31	215.50	155.39	147.04
Req. From DRAM	313.70	208.03	297.67	221.40	189.00
L2-DRAM (Round Trip)	262.03	138.38	238.04	153.14	135.79
DRAM	217.79	88.87	203.26	118.62	105.50

3.8.2 Effect of Mem-Side Caching

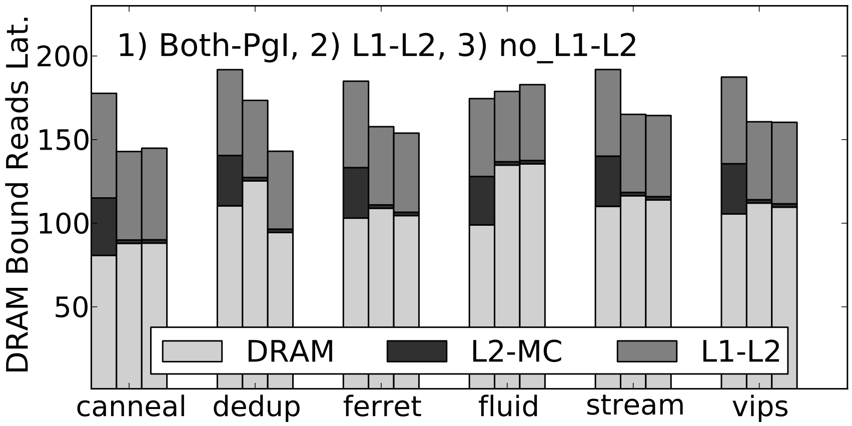


Figure 18: Latency distribution of mem-side cache organization

Figure 18 plots the latency distribution of DRAM bound reads with various memory organizations. The first bar (Both-PgI) represents the case, where both CAM and GAM have been assigned the same mapping functions. The 2nd and 3rd bar (L1-L2 and no_L1-L2) represents our memory-side cache organization with and without the L1-L2 link, respectively. It can be seen that the organizations with a direct L2-MC connection almost removes the L2-MC latency, (It is equal to two cycles, one cycle both ways). Thus, even with higher load on individual DRAM vaults, the overall latency is reduced by 11.7%. fluidanimate is an exception, in which the increase in DRAM latency is high enough to surpass the advantages of reduced L2-MC latency. Furthermore, it should be noted that removing the

L1-L2 link does not significantly increase the latency of DRAM bound reads except dedup. We selected the configuration without the L1-L2 link as our improved memory-side cache organization.

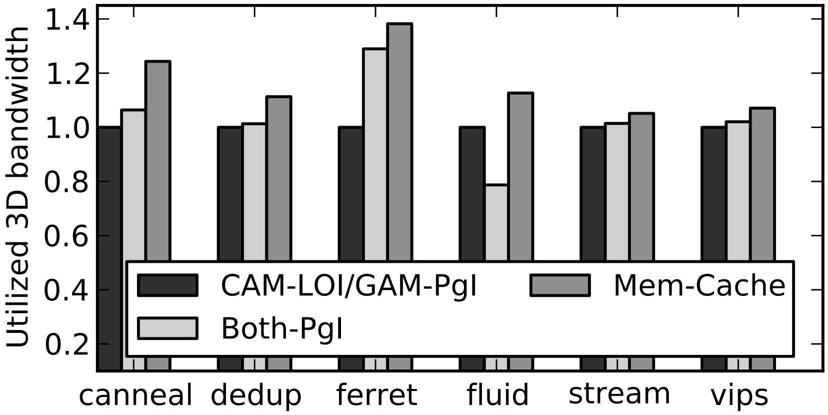


Figure 19: 3D bandwidth utilization normalized to baseline

Figure 19 presents the memory bandwidth utilization (bandwidth across the DRAM channels) of various schemes normalized to the baseline CAM-LOI/GAM-PgI case. It can be observed that in both the cases (that is, applying same mapping for CAM and GAM and the memory-side cache organization) average 3D bandwidth utilization has increased. The increase of Mem-Cache and Both-PgI from baseline is 16.4% and 3.1%, respectively. Small decrease in the case of Both-PgI with fluidanimate can be attributed to the increased DRAM latency (Figure 18), which results in the reduction of its 3D bandwidth utilization.

3.8.3 Neighbor Mapping - Results

Figure 20 shows the latency distribution of reads destined to DRAM for neighbor mapping vs. the case in which no neighbor mapping is used. The first bar represents the case without any neighbor mapping for both CAM and GAM. The second bar represents the case where neighbor mapping is used for both CAM and GAM. The 3rd bar represents the case when neighbor mapping is performed only for GAM. All these cases performed neighbor mapping without the memory-side cache organization. The last bar added neighbor mapping for both CAM and GAM on top of the Mem-Cache organization.

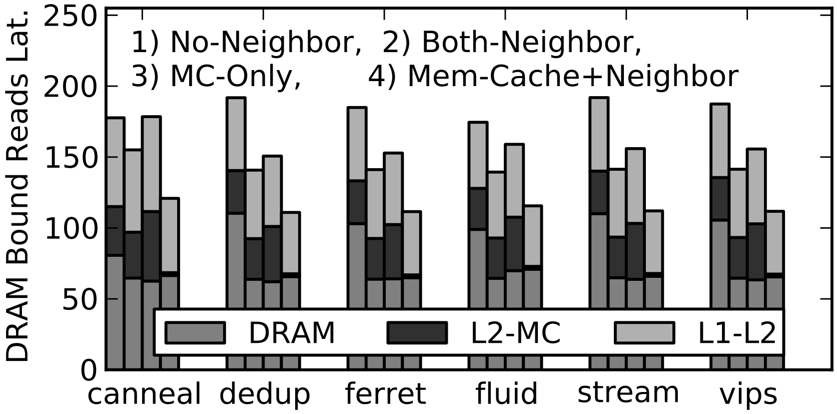


Figure 20: DRAM bound reads latency distribution with neighbor mapping

In all neighbor mapping cases, the queuing latency within the DRAM is reduced. This indicates a good distribution of traffic into neighboring DRAM vaults that reduces load on one specific DRAM. However, the network latency (latency between L1-L2 and L2-MC) has increased significantly for the MC-only neighbor mapping case. This indicates high load on the router that first sends request to L2 and then immediately sends them to neighboring MCs making them a hot spot which resulted in large increase in latency. Finally, the case with neighbor mapping on top of Mem-Cache organization further reduces the latency with no L2-MC traffic. The IPC increase (not shown) averaged across all applications with neighbor mapping on top of Mem-Cache organization from the case with same CAM and GAM mapping is 11.3%, while the increase from Mem-Cache only without any neighbor mapping is 3.5%. Recall that this is in addition to the cases that do not have same CAM and GAM. We conclude that if the distribution of load among different vaults is required, it is better to provide this distribution at the L2 bank level and keep CAM and GAM the same, thus removing any L2-MC traffic in the horizontal direction.

3.8.4 First Touch Policy - Results

Fig 21 presents the global IPC of various benchmarks with and without the first-touch policy. The round-robin policy sequentially assigns pages, which with pgI means one page for each MC in a round-robin manner. It can be seen that first-touch improves IPC for most of the benchmarks. The average improvement over all the benchmarks is 9.6%. This improvement is attributed to the decrease in the average number of hops traveled per response, which is reduced by 5.8% (not shown).

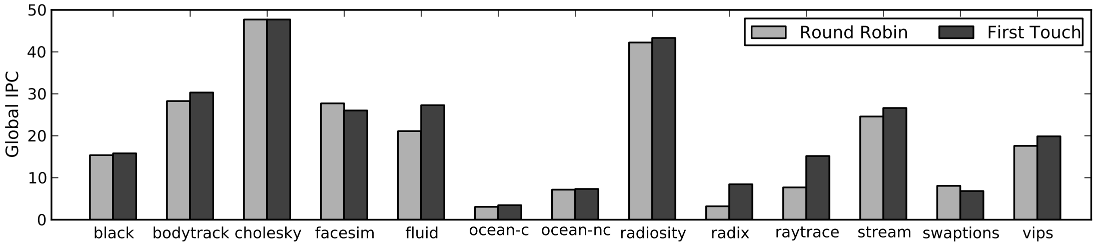


Figure 21: Global IPC with and w/o first touch policy

3.9 Concluding Remarks

This chapter analyzed the performance of an exemplar 3D memory system and proposed a memory subsystem re-organization that places L2 cache banks next to DRAMs with an interconnection network only between the L1 and the L2. It only explored reducing network and DRAM latency by memory reorganization and traffic distribution but did not consider changing the topology or improving various policies at different stages of the memory hierarchy. The following chapter tries to reduce the network latency by designing a low-cost, low-latency router micro-architecture that can be used with high-radix networks to reduce the number of hops traveled by the cache and memory requests, and hence improve overall system performance.

CHAPTER 4

CENTRALIZED BUFFER ROUTERS

The previous section reduces hop counts of the requests without optimizing the network itself. In general, hop count of a network can be reduced by employing high-radix topologies. However, larger radix requires increased wiring and buffering area. Although, wires are abundant on-chip, buffers are a scarce resource and takes significant amount of area and power. Thus most designs end up using low-radix topologies. In this section, we propose the use of centralized buffer routers (CBR) in on-chip wormhole networks to decouple the required buffer space of each router from its radix. Further, we propose to use CBRs in conjunction with recently proposed elastic buffered (EB) links [59] making it feasible to be used with long wires present in high-radix networks. At low loads, CBs are power-gated off and packets bypass them taking two cycles only. At high loads, flits are streamed through the buffered path taking four cycles. We further propose lookahead switch allocation which reduces these paths to one and three cycles, respectively. A novel extension to bubble flow control is used to realize deadlock freedom. The *same* mechanism avoids both routing as well as message dependent deadlock using a constant CB size per router independent of the number of message types. The result is a compact, energy and area efficient physical channel router whose low load performance approaches that of buffer-less routers and high load performance approaches that of buffered routers. This chapter describes the baseline router micro-architecture in detail along with the associated performance and power optimizations.

4.1 The Baseline Centralized Buffer Router

4.1.1 Motivation

The state of the practice for baseline network-on-chip (NoC) routers has been the use of edge buffers, whose buffer capacity requirements are proportional to the router radix and link length (to fully utilize the link in the presence of flow control delays). These buffer requirements are commonly increased through the use of virtual channels (VCs) to ensure deadlock-free routing, and further increased multiplicatively with the number of message classes, to avoid protocol deadlock [19] [34]. This results in substantial area and power devoted to buffers, up to several hundred KBs of storage for a 32-64 node NoC. These overheads mitigate the advantages of NoCs, specially with high-radix routers (which have low hop count and utilize the increased wiring density of NoCs more effectively). We propose a router micro-architecture that effectively reconciles this trade-off between radix and buffer space, using a novel combination of flow control and buffering strategies. The key idea is to use shared central buffers coupled with novel extensions to bubble flow control that reduces the total buffering requirement of the network reducing its area and power while maintaining low-latency, low-throughput operations. The router specially supports high-radix topologies that can be used to reduce the network latency even further. Next, we discuss the micro-architectural details of the centralized buffer router.

4.1.2 Router Micro-architecture:

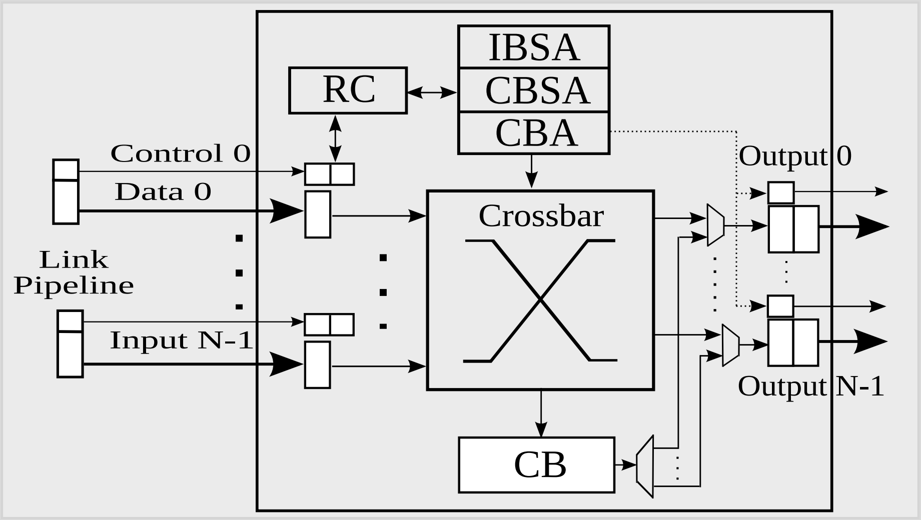


Figure 22: Centralized buffer router (CBR) - Micro-architecture

Figure 22 shows the internal router design. It consists of a large crossbar with single flit input and a 2-flit output staging-buffer for each port of the router. It also consists of a centralized buffer (CB), which is only utilized when a packet from the input buffers cannot progress to the corresponding output buffers. The control and data information are split in the links. This separation will be explained in section 4.2. Central buffer allocator (CBA) performs allocation to the central buffer, while input buffer switch allocator (IBSA) and central buffer switch allocator (CBSA) performs output port allocation for input buffers and central buffers, respectively. The central buffer is a DAMQ [101] style multi-ported output queue in which flits destined to different outputs are kept separate from each other. It can be considered as multiple output queues (one per port) which share each others space. We kept the number of read and write ports of the CB to one, more details of which will be explained in section 4.4.

4.1.3 Pipeline Stages

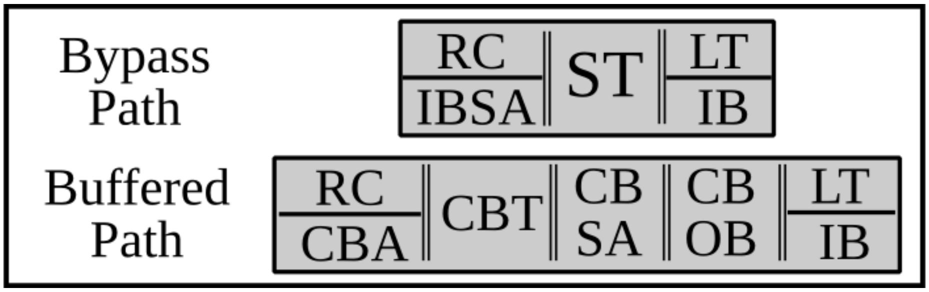


Figure 23: Centralized buffer router - Pipeline stages

Figure 23 illustrates the different pipeline stages of the router. A flit or packet entering the input buffer can take two different paths. 1) Bypass path consisting of IBSA (the switch allocation stage for flits in the input buffer) and the ST stage, and 2) the central buffered path with four pipeline stages within the router; allocation (CBA) and traversal (CBT) to the central buffer, and allocation (CBSA) and traversal (CBOT) from the central buffer to the output port. In both the cases, lookahead routing [22] is used which perform RC in parallel with IBSA or CBA. At low loads, path 1 will be chosen. If the corresponding output port is busy servicing another packet from a different port, path 2 will be selected. Since flits within a packet need to arrive in order, if a path is chosen by the head flit, all subsequent body and tail flits will follow the same path. Furthermore, since interleaving of flits of different packets is not allowed, once an output port is picked by either the CB or any of the IBs, it is not released until the whole packet is traversed. Every cycle, three allocation operations (IBSA, CBSA and CBA) are performed simultaneously. The IBSA tries to allocate a flit at an input buffer to the output port, and if granted, set the necessary crossbar and mux signals. The CBSA, in the mean time, will try to allocate a flit in the central buffer to the corresponding output port. Among the two allocations for the output port, CBSA is given a higher priority, since packets in the CB arrived earlier than the packet in the IB stage. In parallel to these allocations, CBA will also try to allocate central buffer space to packets in the input buffers (one packet for each input port simultaneously). A packet will be allocated to a CB only if the CB has enough space to hold the complete packet. However, if IBSA wins in allocation, CBA will be ignored. Based on which allocation wins, one or two of the three traversals will be performed in the next cycle.

4.2 Lookahead SA

Baseline CBR encapsulates an EB router reducing the input buffering requirements. However, since allocation and traversal are two different stages, the minimal buffering requirement is two flits for 100% link utilization. We further reduce the input buffers to single flit by performing the switch or CB allocation (IBSA/CBA) in parallel with the last LT/IB stage. This will also reduce the latency within the router to one cycle only. Performing allocation in parallel with IB is achieved by separating the data and control information of a single flit and sending the control information one cycle ahead of the data. Note that lookahead routing decides the output port of the next router in the previous one and sends this information along the data (other control information includes flit type, etc). If we can forward this information one cycle ahead of the actual data, e.g., during the ST cycle, it will reach the downstream router earlier, allowing it to contest for allocation one cycle before the data arrives in the input buffers. Thus, when a flit reaches its downstream input buffer, it will perform the ST or CBT stages immediately in the next cycle without waiting for the IBSA or CBA stage to complete. Since route computation can also be done in parallel with allocation, we can again send the next router output port information during the ST stage, that is, one cycle ahead. Note that this is different from prediction router [56], as allocations are deterministic and not predictive. Also, note that the flit control information is already sent out-of-band in most on-chip routers. Even if it is sent in-band, it can be sent with the flit information of the previous flit. Thus, there is no extra wiring overhead of this scheme. We will assume out-of-band control information in this paper.

4.2.1 Guaranteeing one cycle lookahead

A problem with lookahead SA is to guarantee that the control information always arrive one and only one cycle ahead of the corresponding data. This is a necessary condition because if SA wins earlier than the actual data arrival in the input buffer, a dummy flit will be propagated forward from that buffer. Note that in general, this is not guaranteed because data and control can get misaligned along the pipeline.

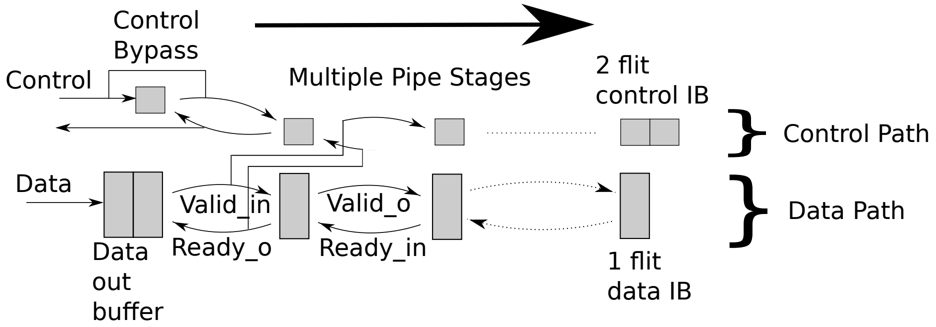


Figure 24: Guaranteeing one cycle ahead - Ready Valid handshake signals of data and control path

We achieved this goal by utilizing the ready-valid handshake signal of the previous pipeline stage in the data path to traverse the next pipeline stage in the control path. This can be seen from Figure 24. The ready_out signal of the data path is also routed to the ready_in signal of the same pipeline stage in the control path. Similarly, the valid_in signal of this pipeline stage in the data path will be sent to valid_in of the next pipeline stage in the control path. This will ensure that once the control information is in the first pipeline stage of the link and the corresponding data is next to leave the output buffer, both will progress across the link, with control information always moving one cycle ahead. However, since the data output buffer can be of multiple flits, ensuring this condition itself requires small control output buffer as shown in Figure 24. When the output buffer is empty, the control information is directly sent to the first control pipeline stage. If the output buffer is not empty, the control information is sent to the control output buffer. The control output buffer, which is one flit smaller than the data output buffer, keeps sending control information corresponding to the second last flit of the data buffer to the first pipeline stage of the link in synchronization with the last flit of the data buffer, thus ensuring that control information always remain one and only one cycle ahead.

At the input side of the downstream router, it is possible that an allocation operation is not successful. Since the data in the last pipe stage of the link sees an empty slot in the input data buffer, it will move forward, resulting in the flit behind it in the link to move forward as well. The corresponding control information also needs to progress to the input control buffer, which already contains the control information of the mis-allocated flit. This means that it is necessary for the input control buffer to be of size one flit larger than the data flit buffer. In the case, a control flit does not win allocation, it should still move ahead in its buffer, allowing the control information of the flits behind it to move forward as well. When the input data buffer is full, the flit in the last pipeline stage of the link cannot progress as the ready_in signal will not be asserted, the control path will stop as well. As long as the flit occupies the input buffer, the corresponding control flit resides in its buffer. As soon as the data flit leaves, its corresponding control information also expires.

4.3 Deadlock Avoidance

As mentioned earlier, our design is similar to wormhole-based router with pipelined links. Therefore, the problem of deadlock avoidance boils down to guaranteeing deadlock freedom in flit level wormhole routers without using VCs. Note that VCs have been a de-facto choice to avoid deadlock in such networks but EB channels create dependencies between flits of two different VCs within the channel. Thus, VC-based approaches cannot be used as a generalized solution. A simple approach to solve this problem is to use strategies like up-down routing that ensures that no cycles occur in the network. [74] recently proposed TRANC routing that uses similar concepts to provide deadlock free routing for multi-dimensional torus networks without using VCs which can be used here. However, a fundamental problem with any such approach is that they do not guarantee minimal paths. This may result in an increase in no load latency. On the other hand, our design extends the bubble flow control technique which guarantee deadlock avoidance (for both routing and message dependent deadlocks) using minimal paths, thus also ensuring minimum no load latency. We have evaluated this scheme with non-minimal TRANC routing in our results section.

Before explaining our scheme, we like to reiterate the three conditions required for bubble flow control to work. 1) Every ring or cycle must have a bubble 2) If there is a bubble in the ring, packets within the ring cannot wait indefinitely on any other condition within the ring, that is, they have to make progress. 3) External packets entering the ring are not allowed to destroy the bubble.

4.3.1 Avoiding Routing Deadlock

The idea of avoiding routing deadlock is simple. For every ring, even having a single flit bubble is enough to ensure forward progress of flits. For flits entering the ring, we need to ensure that all flits of the packet will be allowed to enter the ring when a packet start entering a ring, while maintaining the original bubble of the ring. This is a necessary condition because of the following reason. If the whole packet is not allowed to enter the ring, even having a multi-flit bubble in the ring, e.g., in the input buffer, will not guarantee forward progress, that is, condition 2 above will not be satisfied. This means that a bubble of packet length+1 is required when changing dimensions to ensure deadlock freedom. This bubble can be provided with the output-based CBs without increasing the size of the input and the output buffers. Furthermore, since packets in the central buffer of the current router are part of the overall ring (corresponding to that output port), looking at the space of the next router's CB (which will require CB credit information to flow upstream) is not required. This is because if there is enough space in current router's CB, it is guaranteed that flits from the previous routers will move forward to create at least an equal amount of space in the next router. Thus the condition to avoid deadlock only requires looking at the empty buffer space of itself and no credit information of the downstream router is needed which makes our technique perfectly suitable for EB-based channels. A formal proof is given below.

Definitions: These definitions are derived from [16], [17]. Let Q be the set of input, output, and link pipeline buffers associated with the routers. For each $q_i \in Q$, let $cap(q_i)$ be its maximum capacity in flits and $size(q_i)$ be its current occupancy. A bubble of X flits in q_i means that $size(q_i) <= cap(q_i) - X$. Let Q_y be a subset of Q consisting of all input, output, and link pipeline buffers that belong to a ring y. Let $q_i \rightarrow q_j$ defines the case when a flit form q_i moves to q_j. To avoid interleaving of flits of different packets, if a head flit from buffer q_i moves to q_j, q_i will hold q_j until the whole packet is transferred, that is, no other buffer q_k can move any flit to q_j. Let L be the size of all the packets.

Rule 1: A unidirectional ring y is deadlock free as long as there exists a single flit bubble in Q_y, that is,

$$\exists q_i \in Q_y : size(q_i) <= cap(q_i) - 1. \tag{1}$$

Proof: The rule is a direct consequence of Theorem 1 in [16]. In that paper, the minimum bubble size is equal to the size of the input queues. However, this is a necessary condition for adaptive routing schemes, where head flits cannot make progress if the downstream input queue is not free. Since, we are only dealing with deterministic routing, this condition gets relaxed to a single flit in the downstream input queue and hence the bubble size of one flit. □

Rule 2: If a head flit of a packet A from q_i in the ring x wants to move to q_j in ring y without causing a deadlock, it can do so only if,

$$\sum_{\forall q_k \in Q_y} size(q_k) \leq \left[\sum_{\forall q_k \in Q_y} cap(q_k) \right] - L - 1 \;\; \land \;\; \nexists q_m \in Q_x : q_m \rightarrow q_n \in Q_y, \forall Q_x \subset Q. \quad (2)$$

i.e., a free space of packet length+1 is available in the ring and there is no other flit of packet b entering the ring y at the same time.

Proof: Multiple cases exist. (i) One monolithic bubble of length $L+1$. In this case, this bubble will flow back such that q_j will get a bubble. Head flit from q_i will move to q_j in ring y and the remaining bubble length will be L. Since no other head flit or packet can enter ring y (bubble length is smaller), only flits of packet A can enter the ring. $\sum_{\forall q_k \in Q_y} size(q_k)$ after all flits of A have entered the ring is $\left[\sum_{\forall q_k \in Q_y} cap(q_k)\right] - L - 1 + L$, i.e., bubble length of 1 will still be there. Rule 1 above will hold and allow ring y to be deadlock free. (ii) If multiple smaller bubbles are available with aggregate equal or more than $L + 1$. This means that q_i sees a bubble of length smaller than $L + 1$, lets say l. However, at least one more bubble exists in the ring apart from the bubble in q_j, i.e., $\exists q_m \in Q_y \backslash q_j : size(q_m) <= cap(q_m) - 1$. Rule 1 implies that flits in upstream buffer q_n will move forward to q_m creating bubbles in q_n. Let q_m denote the new buffer with the bubble. The previous process continues, until $q_n \equiv q_j$. Since the number of such bubbles is $L + 1 - l$, the resultant monolithic bubble in q_j will be equal to $L + 1$. Case (i) above is applied. (iii) Bubble length is less than or equal to L. If it is equal to L, after all flits of A enter ring y, $\sum_{\forall q_k \in Q_y} size(q_k) = \sum_{\forall q_k \in Q_y} cap(q_k)$. No bubble exists in ring y, rule 1 will not be satisfied any more. If the bubble length is less than L, whole packet A is not allowed to enter ring y. Since, incomplete packet traversal means q_i hold q_j, no other buffer q_k will move flits to q_j even if there are bubbles in the ring. Similarly, no other buffer q_m will be able to send to q_k and so on. Thus no flit will be able to move forward resulting in the ring being deadlocked. (iv) If any other packet is allowed to enter ring y, simultaneously with packet A, it will reduce the bubble size resulting in case (iii) above. □

Implementation of rule 2 is difficult, since it requires global information to restrict packets from injection into the ring based on whether any other packet is being injected at the same time. A much simpler local condition is to check the bubble in the central buffer of the local router. Suppose that the central buffer reserves 1 packet for each ring. Let Q_{y2} be the set of reserved packet spaces for ring y in all the central buffers corresponding to nodes in ring y. Thus, Q_{y2} becomes part of the new extended ring y. Let Q_{cy} be the union of Q_y and Q_{y2}. Above 2 rules can be applied to the extended ring.

Rule 3: Rule 2 can be satisfied by the following 3 conditions as well. (i) Look for a space of L in local central buffer corresponding to ring y, (ii) Look for a space of 1 flit in the output buffer of ring y, and, (iii) Not allow any other packet of the local router to enter ring y at the same time.

Proof: The proof is straight forward, since having bubble of $L + 1$ in the local router for extended ring y satisfies equation 2 with Q_y being replaced with Q_{cy}. Also, since we are only looking at the local router, not allowing any other packet to enter ring y at the same time is straight forward. This also means other packets can enter ring y in other routers. Case (iv) of the proof of rule 2 will never happen with flits entering the ring in other routers. □

Reiterating the minimum deadlock condition:

$$FreeSpace = \begin{cases} 1 & i = j \\ PktLength + 1 & i \neq j \end{cases}$$

where i, j refer to different dimensions of travel. We have used empty space of *PktLength* in CB and space of 1 flits in corresponding OB. This condition is checked during allocation of both OBs and CBs, that is, during IBSA and CBA to ensure a bubble is maintained in the ring. Furthermore, checking full packet space is not required during CBSA as the packet has already entered the ring and therefore, same dimension condition will be applied here. This makes the minimum CB buffer size requirement to $2 * dim * PktLength + 1$ flits. In practice, we can reduce the CB size to $2 * dim * (PktLength - 3) + 1$ by leveraging the fact that the 2-flit output and 1-flit input buffer is part of the overall ring. We fixed the size of CB to 18 flits. Starvation is also possible with CBs. We ensured starvation does not occur by round robin allocation of central buffers to each port. We would also like to mention here that this solution is feasible for on-chip networks where packet size is not large. In fact, all bubble flow control techniques except worm-bubble [16] are not good solutions for networks with large packet sizes. Variable packet sizes are allowed as long as each ring keeps a bubble of the maximum packet size.

4.3.2 Avoiding Message Dependent Deadlock

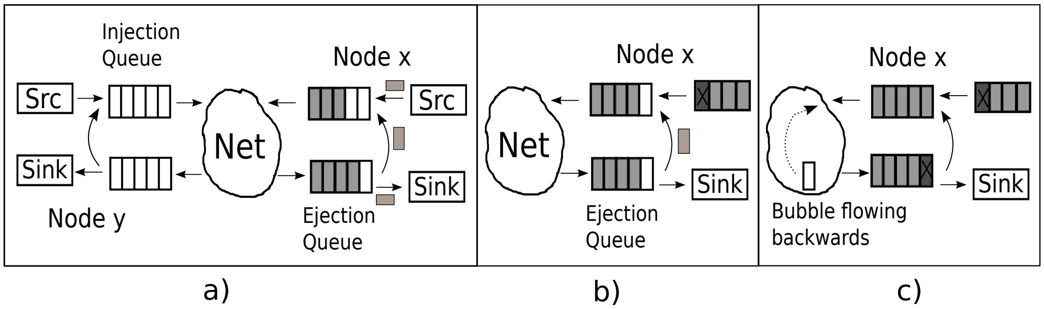


Figure 25: Message Dependent Deadlock

Message dependent deadlocks are usually solved by providing separate virtual networks as explained in [34]. The basic idea is to provide separate virtual network (channels) to every class of a message dependent chain. We propose the use of bubble flow control technique to avoid message dependent deadlocks as well. Note that every reply of a request message (e.g in a request reply network) can be considered as a 180 degree turn of the same packet, allowing the possibility of cycles between 2 or more different request-reply pairs. The packet source injecting new requests can be considered as an external entity that inserts new packets in this cycle, (Figure 25(a)). These cycles will be deadlock free as long as we ensure that the three conditions of deadlock avoidance mentioned earlier are satisfied.

Condition 1 and 3 can easily be satisfied by inserting request messages in the injection queue of the network interface (NI) only when there is a space of at least two packets, (Figure 25(a)). Satisfying condition 2 means that if there is only one empty space left in the injection queue, the reply message of a request will still be generated, even if there is a new request message pending to get inserted in the injection queue, (Figure 25(b)). Thus a packet present within the message cycle (request message in the ejection queue) is allowed to progress to the next buffer (generate a reply message) with space of only one packet downstream, i.e., in the injection queue. External packets (pending request messages in the message sources) have to wait. Implementation of this scheme in network terminals means having the ability to accept new request messages and generate the corresponding reply, if there is an empty space in the injection queue. If it has no empty space, (Figure 25(c)), request messages can wait in the ejection queue but since there will be a bubble in the message cycle somewhere, this bubble will always propagate back to the injection queue allowing the request messages to get serviced. Note that this scheme is valid for any number of message classes without adding VCs.

The use of bubble flow control in the preceding manner makes it possible to deal with routing deadlock and message dependent deadlock with the *same* mechanism, e.g., there is no need for additional storage such as separate request and reply networks. In particular, the cost of dealing with message dependent deadlock is fixed independent of the number of message classes. Overall, the cost of deadlock freedom at a router is independent of the network size or the number of message types.

4.4 Power Reduction Techniques

CBR coupled with lookahead SA reduces the input buffering requirements of a router to only a single flit. However, it increases the area of two more components; 1) it increases the crossbar size, and 2) it adds the central buffer space. Power advantages gained by removing the input buffers are reduced significantly by the addition of these components. We applied two simple techniques to reduce the overall power of the routers.

4.4.1 Reducing the number of CB ports

Since the utilization of the central buffer is low (only used at high throughput), we can reduce the number of its input ports and allow them to be shared by different inputs. We reduced the number of input ports of the CB to only one. This means that if two or more packets at the input buffers have to traverse to the CB, they will be serialized. The CBA stage ensures that it allocates only one flit to the CB each cycle. This approach not only reduces the input ports of the CB, but will also reduce the output ports of the crossbar. The new crossbar is only $Inports \times (Outports + 1)$ switch with much reduced area and power dissipation. The same principle is applied to the output ports of the CB as well. Again, this implies that if two flits in the CB want to reach the output buffers in the same cycle, they need to be serialized by the CBSA, even though they belong to different output ports of the router. The performance overhead of this serialization is small (since CBs are utilized seldom and do not require high throughput), however, the power reduction by reducing the ports is significant.

4.4.2 Power Gating of CB

Since CBs are utilized only at high loads, we applied a simple coarse grained power gating technique to it. Power gating of CB is simplified as it does not interfere with the main path of the router. Deadlock avoidance will be guaranteed as long as it turns on in some finite amount of time. Initially, the CB is kept off. Whenever two packets collide for an output port, a counter starts counting the wait cycles of the unallocated packet. When the wait becomes X cycles, the CB is turned on. We assume that it takes three cycles for the CB to turn on completely. Once on, unallocated packets can be pushed into it allowing the blocked packets to move ahead. When the CB remains empty in the on-state for a minimum on-time (empirically set to be 10 cycles), it is turned off. At low loads, this simple power gating technique keeps the CB turned off, saving power. At high loads, since we wait for X number of cycles before turning it on, this technique can potentially reduce performance. In fact, the value of X provides the throughput power consumption trade-off. Greater the value of X, lesser will be the power as well as the saturation throughput. The sensitivity of the value of X is explored in section 4.5.

4.5 Results

Next, we compare the power and performance of our centralized buffer routers with different state-of-the-art solutions under different traffic conditions and topologies.

4.5.1 Simulation Setup

We have developed four different router micro-architecture models to understand the latency and throughput impact of our centralized buffer router design. The baseline router consists of a standard 2-stage pipeline router with two VCs per virtual network. The other two routers implemented are 2-stage EB router and a simple flit deflection (FD) router similar to flit BLESS from [61]. Parametric configurations of each of the routers is given in Table 7. The DAMQ based central buffer is organized into six slots of 3 flits each. The default wait time before turning on the power gated CB is 500 cycles. Furthermore, to reduce the latency and buffering requirements of the deflection router, we retire the packets as soon as the tail flit arrives without waiting for head and all body flits to reach the destination. This makes the deflection router very optimistic. Since EB requires duplicate physical channels, we have assumed its links to be half wide with twice the number of flits per packet.

Table 4: System configurations of various routers

Parameter	Baseline	FD	EB	CBR
Pipeline Depth	2	1	2	1
InBuf Size (per port)	5*VC	1	2*Virt. Net	1
OuBuf Size (per port)	2	2	2	2
CBuf Size	na	na	na	18
Inj/Ej Que Size	20	20	20	20

We have implemented mesh, torus and generalized hypercube (GHC) topologies for both 2D and 3D networks. The 2D networks are 8x8 while 3D networks are 4x4x4. Number of ports for mesh and torus are $2 * dim + 1$ and $(k - 1) * dim + 1$ for GHC topologies. Thus 2D-GHC, with $k = 8$ has the highest number of ports and therefore has the highest power consumption. The torus and mesh networks have single cycle link delays between adjacent routers. The GHC models multi-cycle links equal to the number of routers between the source and destination, that is, the link delay for node 2 and 3 from node 0 in the x dimension will be two and three cycles, respectively. The packet size is kept fixed (= five flits) except EB routers which are ten flits as discussed earlier. All links are assumed to be 128 bits wide. All designs except deflection routers use minimal dimension order routing. For torus topologies with single VC and no central buffering, tranc routing from [74] is used which is an up down style non-minimal routing technique that does not use VCs.

Four different synthetic traffic patterns (random, bit complement, bit reversal and tornado) are evaluated; see table 5. Unless otherwise stated, all results present an average of all four as shown in Table 5. Random distributes the traffic evenly and has high throughput. All others are adversarial traffic patterns with relatively low throughput. Tornado travels equal or more than k/2 hops in each dimension and thus has the highest no load latency. All simulations are performed for 50 million cycles. Applications traces are taken by running 64 threaded version of PARSEC and SPLASH benchmarks with 64 cores, 16 MC configuration using an in-house simulator with DRAMSim2 [81] as the main memory model. The traces are generated at the back side of L1 and messages are classified into read/write/coherence type requests. A reply of five flits is generated from the destination every time a read request is received. Read requests and coherent messages for all networks including EB consists of two flits and write messages are five flits except in the EB network in which they are ten flits wide. This allows us to test our scheme for variable size packets as well.

For power modeling, Orion 2.0 [111] is used which calculates the router power as the sum of the power in its buffers, crossbar, arbiters, and allocators along with the link power. We modified Orion to get more accurate results. As a conservative estimate, EB links are

Table 5: Various traffic patterns

Traffic	**Description**
Random	destination chosen with uniform distribution
Bit Complement	(id) to (NODES-id)
Bit Reversal	(x5,x4,x3,x2,x1,x0) to (x0,x1,x2,x3,x4,x5)
Tornado	(x,y,z) to ((x+k/2)%k, (y+k/2)%k, (z+k/2)%k)

modeled to take 3x more device power and 3x more leakage power in routing logic than non-EB links. Similarly, the CBR which has three arbiters takes 3x more power in arbiters. VC allocator power is assumed to be negligible for all cases. Segmented crossbars with two segments are used. For GHC topologies, partitioned crossbars are used. Baseline and EB routers are assumed to have two message classes, with two VCs per message class. FD and CBR do not model any VC or message class. CBR has an additional component of power due to its central buffer. All buffers are assumed to be register based. The network is modeled to be running at 2.0GHz with Vdd = 1.0V and 45nm technology. Activity for different components such as crossbar and input output buffers etc. are taken directly from performance simulations and fed as activity of different components of Orion. Power gating a CB is assumed to reduce its leakage to 20% of the original.

4.5.2 Performance with Synthetic Traces

Comparison with Other Routers: Figure 26 compares throughput and average packet latency of CBRs with that of other routers with different network configurations. Note that the throughput is defined as retired flits per node per cycle. The results presented are aggregated over all four traffic patterns. At low loads, CBR network has the latency equal to that of deflection (FD) router. This is because of the single cycle latency within the router. Both baseline and EB has two cycle latency within the router resulting in increased no load latency. It should be noted that pipeline bubbles are avoided in these designs by keeping large buffers at the input. Furthermore, EB has higher serialization latency since each link is narrower than the other routers.

Baseline and deflection routers have the lowest saturation throughput. For deflection

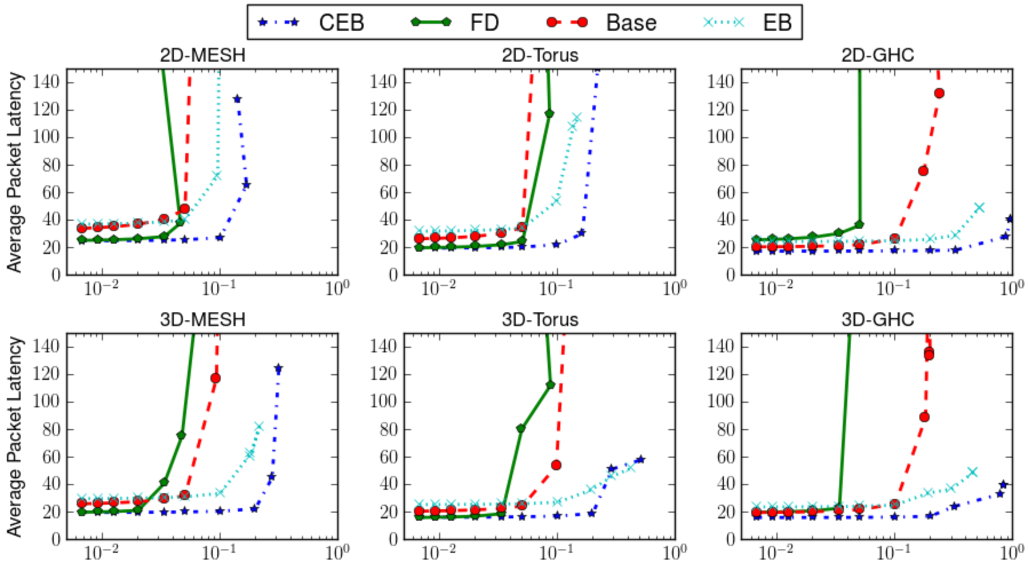


Figure 26: Throughput (retired flits / node /cycle) vs average latency (cycles) for different network configurations

routers, the greater the number of ports per router, more numerous the deflections are, and thus saturation throughput does not increase with the number of ports. This can be seen in the case of GHC, where deflection router saturates quickly compared to other routers. The baseline router has higher throughput than deflection in most cases, but because of extra bubbles created due to credit-based flow control, their throughput is low as compared to routers that use elastic links even with multiple VCs. This difference increases with longer links in GHC topologies.

Both EB and CBRs have much higher throughput due to the use of elastic links. CBR has higher saturation throughput due to the removal of head of line blocking made possible through the central buffering. However, since traveling to the central buffer increases latency within the router, this is not always true. Figure 27 shows the performance of 3D torus with individual traffic patterns. Note that for Tornado traffic EB performs better than CBR. Since Tornado is an adversarial traffic pattern, it requires larger number of packets to traverse the central buffer and thus increased latency within the router and lower throughput. A similar behavior can be seen for the 3D Mesh topology in Figure 26, where the CBR curve starts going backwards. This also means that a very high utilization of CB is not desirable as it increases the latency within the router by taking buffered path most of the time.

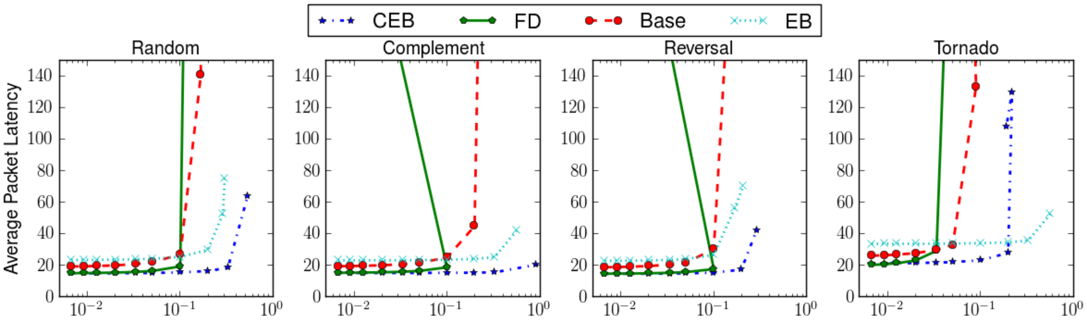


Figure 27: Throughput (retired flits / node / cycle) vs average latency (cycles) with different traffic patterns

Comparing different topologies for CBR, it can be said that greater the number of ports, lower will be the hop count and lower will be the no-load latency with high saturation throughput. This conforms to the fact that the greater the power and area budget available, better will be the performance. This is not true in the case of other routers like deflection and baseline which has relatively slower increase in performance with increase in the number of ports. This makes CBR an ideal choice for large-radix on-chip networks.

Impact of Individual Optimizations: CBR uses various optimizations for different purposes, e.g., lookahead SA for latency reduction, power gating for power reduction, and bubble flow control for deadlock avoidance. We compare the advantages of individual optimizations in Figure 28 for various network topologies. In the figure, NOBUBBLE represents the case without any optimization and no bubble flow control. NOSA adds bubble flow control to the NOBUBBLE case. NOGATE adds lookahead SA to the NOSA case without power gating. It can be seen that NOGATE and GATE cases which have single cycle latency in the router by adding lookahead SA optimization has significantly low no load latency. Their throughput, therefore, is higher in general. The torus topologies with NOBUBBLE have higher no load latency due to non-minimal routing (remember we use tranc routing for these cases). However, the saturation throughput of 3D torus with non-minimal routing is higher which shows the overhead of having bubbles in the network. Note that both NOSA and NOBUBBLE case has two flit input buffer as opposed to single flit in other cases. Lastly it should be noted that power gating closely tracks the case with no power gating specially in the case of GHC topologies, thus its performance overhead is low.

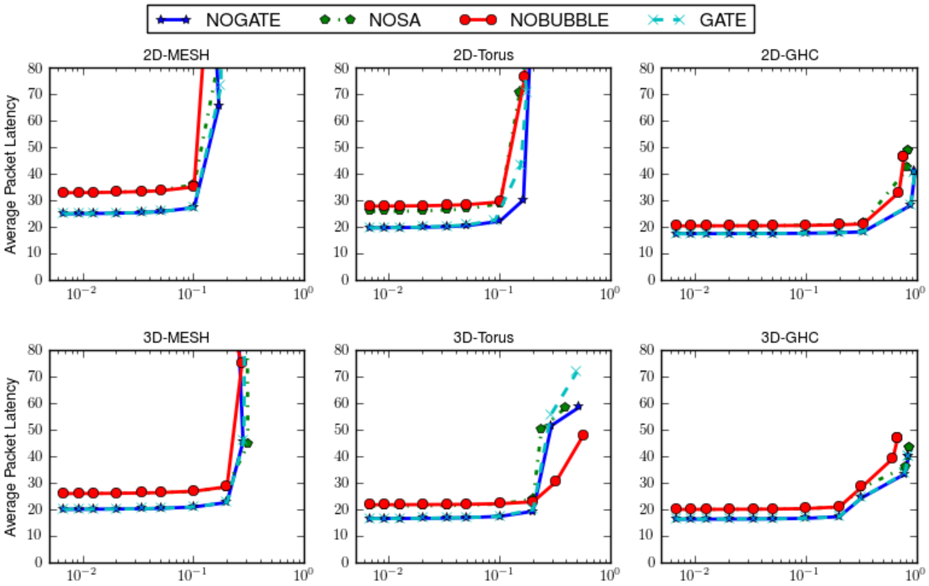


Figure 28: Performance Impact (throughput vs latency) of individual optimizations

Comparison of different topologies: The above figures can also be used to compare the results of different topologies for CBR routers. Note that they have different link bandwidth and buffer requirements and therefore different area and power. The no-load latency of GHC topologies are the lowest. Their saturation throughput is close to one. Mesh topologies have the highest no-load latency due to large number of hops for a request and thus lowest throughput. In general, the greater the number of ports, the lower will be the hop count and lower will be the no-load latency with higher saturation throughput. This is not true in the case of other routers like deflection and baseline which has slow increase in performance with increase in the number of ports making CBR well suited for high radix networks.

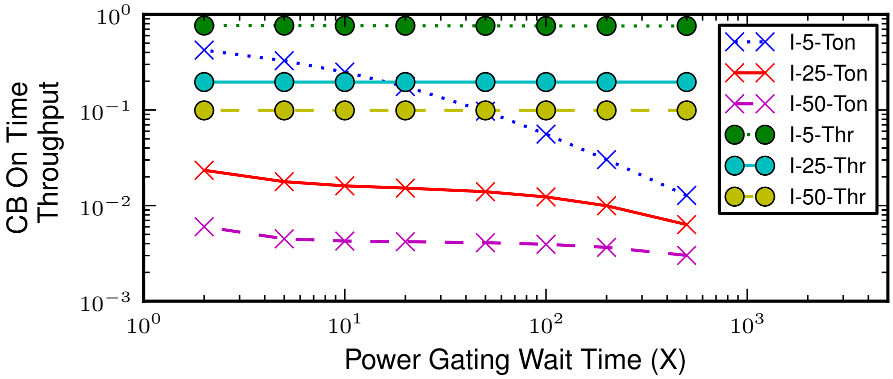


Figure 29: Sensitivity to power gating wait time (cycles)

Sensitivity to Gate Wait Time: As discussed in section 4.4, central buffer is only turned on after waiting for X number of cycles. The wait time should have an impact on both throughput and power. Figure 29 plots saturation throughput (Thr) and the percentage of time CB was on (Ton) by changing the value of X for 2D GHC topology with different injection rates. Here, I-5, I-25 and I-50 mean that the maximum injection time between subsequent packets are 5 (one cycle per flit), 25 and 50 cycles, respectively. We can see that CB on-time greatly reduces with increase in the wait time while the reduction in throughput is extremely small. Thus, large values of X can be used allowing higher leakage power reduction. We fixed the value of X to be 500 in all our power gating simulations. Dynamically adjusting the wait time at different loads can further reduce power reduction and improve throughput.

Table 6: Buffer space (KB) with different configurations

RowNo	Parameter	2D-Torus	3D-Torus	3D-GHC	2D-GHC
1	**Baseline-M1**	100	124	110	145
2	**EB-M1**	60	68	60	70
3	**Baseline-M4**	280	376	320	460
4	**EB-M4**	120	152	120	160
5	**FD-P4**	55	61	70	85
6	**FD-P20**	135	141	150	165
7	**CBR-GATE**	55	61	70	85
8	**CBR-NOGATE**	73	79	88	103
9	**CBR-NOSA**	78	86	98	118

4.5.3 Buffer Space Reduction Analysis

We performed buffer space analysis for different routers and optimizations of CBR. Table 8 gives the total buffer space requirements of different routers with different topologies. The formula for calculating the buffer space is $[(P * (F * VC + O) * M) + C + I + E] * L$, where L is the link width, P is the number of ports per router, F, O, C is number of flits in input, output and central buffer, respectively and I and E are the injection and ejection queue size in flits. M is the number of virtual networks required to support different message classes. For this analysis, torus topologies use two physical channels or two VCs in EB or baseline router, respectively and GHC use one VC.

We can see that the baseline router requires large buffer space even with single message class (row 1). EB with one message class requires less storage but it increases significantly with the increase in number of message classes as can be seen by row 4 with four message classes. Since GHC topologies use only one VC or virtual network, the storage requirement is reduced, however this will reduce throughput as well (not simulated). FD (row 5) requires the least buffering space. However, if we consider that it has to re-organize flits coming out-of-order at the network interface, which requires larger storage, the buffer space requirement of FD will also increase. If we increase the flits space in ejection queue by five times, the buffering requirement of FD easily surpass most other networks (row 6), since the total ejection queue size aggregated over all NIs (which is already 20K) will be increased to 100K. Thus reorganization overhead of FD is high both in terms of buffer space and latency (not modeled).

Baseline CBR requires more buffer space than EB for single message class due to the presence of central buffer (row 8). But since, it does not require extra buffer storage to handle message classes, the storage requirement does not increase. When the gating is turned off, CBR has equal amount of buffer power as that of FD with no re-organization overhead. With gating on, the storage requirement increases slightly, however the increase is small specially for high radix topologies. On the other hand, as discussed earlier the throughput will be extremely high as compared to FD. Finally, the last row shows that lookahead SA saves 5K and 15K of buffer space for 2D torus and GHC topologies, respectively.

4.5.4 Power Analysis

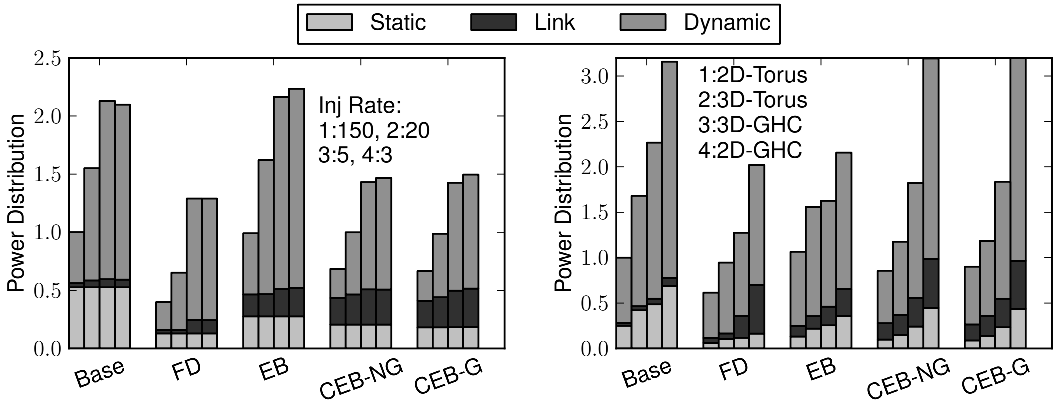


Figure 30: Static and dynamic power distribution (a) varying injection rate, (b) varying topology

Figure 30 (a) compares the static and dynamic power dissipation of different routers configured in a 2D torus topology and normalized to the baseline case at an injection rate of 150 cycles per packet. The different bars represent different injection rates (maximum time between two successive packets from a node) as given by the text in Figure 30(a), that is, bar 1 represents injection rate of maximum 150 cycles per packet, bar 2 represents injection rate of 20 cycles per packet and so on. As obvious, baseline and FD have the maximum and minimum static power, respectively (the bottom component of each bar). Among the routers with elastic links, EB has more static power than CBR. This is because of the minimum requirement of having 4 physical networks (2 for each message class). It also results in EB having the highest dynamic power specially at high loads. Note that EB routers are 64 bit and individual networks have lower power. Elastic links have high power compared to others because of their higher activity and larger unit power (power required to traverse a flit). The dynamic power of CBR is low compared to baseline and EB routers due to its small buffering space. Power gating of the central buffer, although, reduces static power but its advantages at high loads are small.

Figure 30 (b) shows the same plot with various topologies. This time it is normalized to the baseline case with 2D torus topology. CBR has static power increase comparable to EB router. The dynamic power of CBR, however, increases rapidly because of its high saturation throughput and thus high activity. Small increase of dynamic power in EB routers is attributed to thinner channels and crossbars. Although, this along with high saturation throughput makes EB routers a good candidate for NOCs, they loose on no-load latency. Furthermore, their power increases dramatically with increased number of message classes.

4.5.5 Results with Real Benchmarks

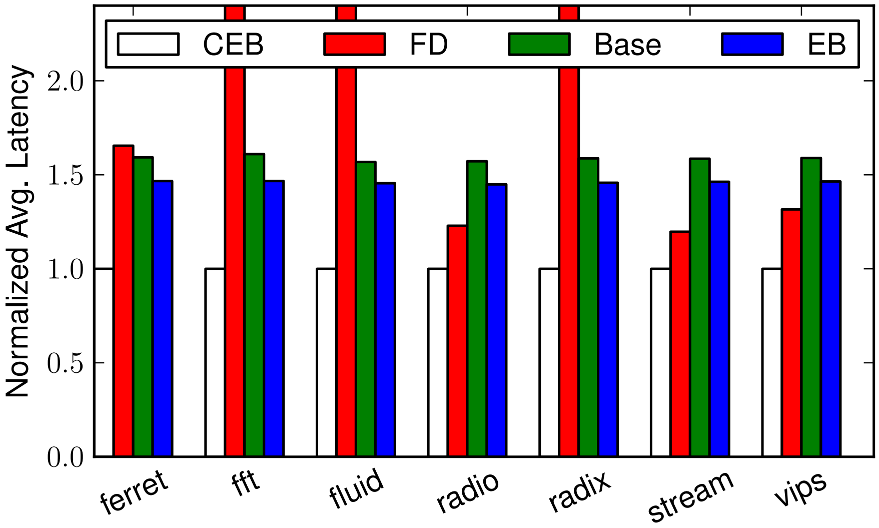


Figure 31: Normalized average packet latency for real application traces

Figure 31 shows average packet latency of 2D-Mesh network with different routers normalized to the CBR case. In general, FD has the maximum average latency while CBR has the least. In few of the benchmarks, this latency is extremely high. This is due to unnecessary deflections and lack of starvation avoidance in FD routers. In these routers, packets at the injection queue are prioritized lower than the packets already present in the network guaranteeing availability of ports. However, this can potentially lead to starvation at very high load and thus increased latency. Average latency of baseline and EB routers increase by 1.4-1.6x than CBR due to increased no load latency and lower throughput. The trends are similar across different benchmarks.

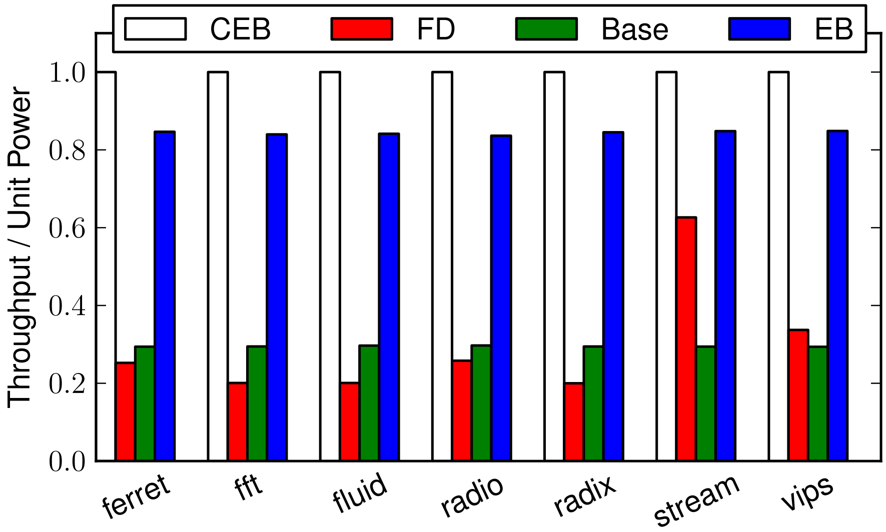


Figure 32: Normalized throughput per unit power for real application traces

Figure 32 shows the throughput per unit power of the same configuration normalized to the CBR case. CBR performs better than all other routers. Again, FD routers perform the worst because of its large packet latency. All other routers have similar throughput as they retired almost equal number of packets in a fixed amount of time. However, the power consumed by baseline and EB router is higher than the CBR case. This behavior is directly attributed to higher latency and larger buffering requirements of both the baseline and EB routers. We conclude that CBR reduces power at fixed load and decreases average packet latency. If further reduction in latency is required, high radix topologies can be used. Under a fixed load CBR will perform better (both in terms of latency and power).

4.6 Concluding Remarks

This chapter presented the baseline centralized buffer router, a low-latency, low-power micro-architecture that is tailor-made for high-radix networks. We showed that a small central buffer in an EB channel-based design can avoid deadlock and improve throughput without the need of having separate virtual channels or physical networks. Virtual channels, however, are also used to provide support for adaptivity and separation of flows for quality-of-service purposes. Furthermore, the minimum size of the central buffers is important to reduce area and power. We will focus on these issues in the next chapter.

CHAPTER 5

IMPROVING DETERMINISTIC CBR - BUBBLE-SHARING FLOW CONTROL

Although CBR provides a shared buffer scheme suitable for high-radix routers, the modified bubble flow control technique still works at a packet level in the central buffers, thus increasing the minimum buffering requirement of the central buffers. In this chapter, we reduce the buffering requirements of the central buffers by introducing *Bubble Sharing*, a flow-control technique that extends the worm-bubble flow control [16] idea to centralized buffer routers. The scheme works at a granularity of a few flit-sized slots called worm-bubbles, reducing the total buffering requirement of the central buffers. We also propose *Adaptive Bubble Sharing* that enables adaptive routing with bubble-sharing flow control for wormhole switched networks. The result is a large reduction in buffer space for both adaptive or deterministically routed high-radix NOC routers harnessing the benefits of high radix while minimizing traditional high buffer space overheads.

5.1 Bubble Sharing with the Central Buffers

This section describes our bubble sharing scheme that extends the ideas used in WBFC to centralized buffer routers. We first explain the key ideas of WBFC using an example and points out the modifications required to adapt it to centralized buffer routers. We organized the central buffer as dynamically allocated multi-queue (DAMQ) [101] with shared pool of small worm-bubble sized slots. Each slot has a space of 2-3 flits and assigned completely to an output port; that is, once a slot is assigned to output *x*, all entries will be consumed by packets going to output *x*, until the slot is unassigned. Each slot act as a worm bubble multiple of which can be taken by each ring (explained later).

5.1.1 Black & White Bubbles:

Consider the example in Figure 33 with edge buffer routers. Suppose each bubble indicates an empty input buffer in the downstream router and is denoted by *worm-bubble* or simply *bubble*. Black means a marked *worm-bubble* and white means an unmarked *worm-bubble*. Let's assume the size of each *worm-bubble* as *x* and packet size as *M*. Let *PktS_WB = M/x* denote packet size in terms of worm-bubbles.

Injection: Before insertion into a ring *x*, a new packet first marks the *worm-bubble* of the corresponding ring as black. This is shown in Fig 33 a), where a packet at R4 is trying to get injected. It marks the corresponding bubble as black, and also maintains a count of the marked bubbles, shown by *CntI = 1*. Forward movement of flits displaces the bubble along with their color backwards. Hence the black bubble at R4 will be pushed backwards to R3, leading to the reappearance of a white bubble at R4. The packet trying to get injected can again mark this bubble as black, further incrementing its count. Now consider the case when *PktS_WB-1* worm-bubbles have already been marked for the ring, (i.e., $CntI \geq PktS\_WB - 1$), and it encounters an empty white bubble. If the packet is injected now, it will occupy the space of bubbles marked by itself and the current white bubble. Hence, guaranteeing that any black worm-bubble injected in the ring during initialization, will remain intact. This is the key idea in WBFC that has been used in our Bubble Sharing scheme as well, using the central buffers. Central buffers, as explained earlier, provide a shared pool of these bubbles, instead of having one bubble for each ring per router. Hence, instead of reserving a black bubble every cycle for a particular ring, multiple black bubbles can be allocated simultaneously. A count, called *WhiteBubbleCnt*, is maintained to keep track of the unoccupied white bubbles. Hence, if $WhiteBubbleCnt + CntI \geq PktS\_WB$ and there is an empty white bubble, injection can happen directly. It should be noted that during injection, *CntI* is passed to the head flit of the packet that maintains a counter called *CntH* to keep track of the marked bubbles by this packet. The rules for injection are given by label 4 & 6 in algorithm 1.

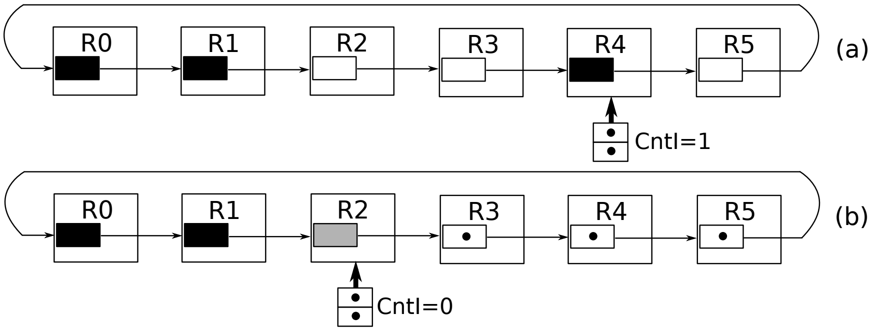


Figure 33: Worm-bubble flow control (WBFC)

Transit: In the original WBFC, the marked bubbles are unmarked when the packet is moving through the ring, decrementing *CntH*. In case of Bubble Sharing, multiple black bubbles for a particular ring are unmarked simultaneously at a particular router. This quickly reduces the amount of black bubbles in the ring leading to a significant reduction in packet injection delays as compared to WBFC, specially in the case of long packets. The rule is given by label 11 & 12 in algorithm 1.

Ejection: If the packet does not encounter an equal amount of marked bubbles, the remaining count is passed to the ejecting router, which means that this router has already injected a few black bubbles into the corresponding ring. This rule, (label 10 in algorithm 1), remains the same in both WBFC and Bubble Sharing scheme.

5.1.2 Gray Bubble:

If multiple ports are marking bubbles in the ring simultaneously, it is possible that all the bubbles in the ring are marked without any packet being injected. This can lead to starvation. A gray bubble was introduced in the original WBFC that allows packets to be injected even if the input port has not marked enough bubbles. An example of this is shown in Fig 33 b), which illustrates a case where all bubbles are either occupied, or have been marked as black except for one gray bubble. Since the packet encounters a gray worm-bubble, it will be injected. To ensure that this packet does not consume all the empty space, *PktS_WB* black bubbles and one gray bubble was inserted in the ring at initialization (label 1, algorithm 1), which will guarantee that even after injection by the gray bubble, at least one original critical bubble remains intact. The gray bubble will be consumed by the packet after insertion (rule 5), which will prohibit any other packet to enter the ring using the gray worm-bubble. When the packet containing the gray bubble will be ejected, it turns the bubble at that port gray, restoring the gray bubble of that ring (rule 7 & 8). Further details of why gray bubbles work can be found in [16]. Bubble Sharing scheme uses the gray bubble rules from the original WBFC without any further modifications (label 5, 7, 8). Furthermore, progressive movement of gray worm-bubble is necessary to ensure forward progress of all the packets entering the ring. Details can be found in [16].

```
Init: for each ring do
    Insert $PktS\_WB$ BLACK & 1 GRAY bubble;
    For each router, assign 1 bubble as BLUE;
    WhiteBubbleCnt = non-assigned worm-bubbles;
end
Injection:
if $WhiteBubbleCnt > 0$ and $CntI + WhiteBubbleCnt \geq PktS\_WB$ then
    assignColor(BLACK); // till CntI == PktS_WB-1;
    HF.CntH = CntI; CntI = 0;
else if $isColor(GRAY)$ and $CntI \geq 0$ then
    HF.color = GRAY; color = WHITE;
    HF.CntH = CntI; CntI = 0;
else if $WhiteBubbleCnt > 0$ and $CntI < PktS\_WB - 1$ then
    assignColor(BLACK); // till CntI == PktS_WB-1;
    don't Inject;
end
Ejection:
if $HF.color == GRAY$ then
    if $WhiteBubbleCnt > 0$ then
        assignColor(GRAY);
    else
        turnBlueToGray;
    end
end
CntI=HF.CntH; HF.CntH = 0; HF.color=WHITE;
Transit:
if $isColor(BLACK)$ and $HF.CntH > 0$ then
    removeColor(BLACK); // as much as possible
else if $isColor(BLACK)$ and $HF.CntH == 0$ then
    bkwdDispl(BLACK);
end
CntI_Logic:
if $CntI > PktS\_WB - 1$ then
    bkwdDispl(CNTI); CntI–;
end
```

Algorithm 1: Bubble Sharing Flow Control

5.1.3 Blue Bubbles:

It is possible that one ring takes all the white bubbles and starves others. Moreover, the movement of this ring can be dependent on movement of other rings, for example in xy routing, movement of rings in the x-dimension are dependent upon successful movement of rings in the y-dimension. If the ring x takes all bubbles at a particular router, ring y may not be able to move through that router, which will stop ring x from moving as well, causing deadlock. The example in Fig 34 a) explains the situation. A packet in R2 in ringY wants to move through R6, but all bubbles of R6 are taken by ringX. Thus ringY cannot move, although it has sufficient bubbles in R10. Since ringY cannot move, a packet at R5 in ringX that wants to eject into ringY cannot make progress.

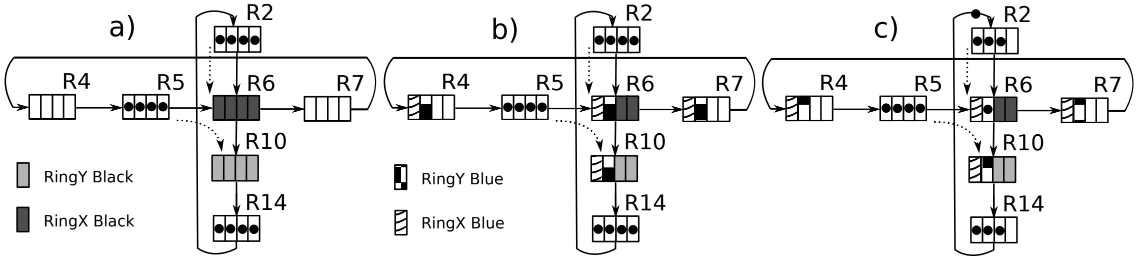


Figure 34: Avoiding one ring to take all bubbles

The problem can be solved by introducing a blue bubble for each router in each ring. This bubble acts as a normal white bubble assigned to that ring, but as a black bubble assigned to a different ring for all other rings. This ensures that at least one worm-bubble slot is kept in each router for each ring, thus allowing that ring to always make forward progress. In the previous example, Fig 34 b) shows blue bubbles for both rings. The blue bubble of ringY will allow flits waiting at R2 to move forward into R6 consuming the blue bubble (Fig 34 c). However, as soon as the flit leaves the router, the blue bubble for that ring is reclaimed, ensuring forward movement all the time. It should be noted that to guarantee progressive movement of a gray bubble, the blue bubble should be converted to gray bubble for that ring, in case no other white or black bubble for that ring is left in that router. This also holds true in the case of ejection. The ejection rule is slightly changed to encounter the blue bubble as shown by label 9 in algorithm 1.

5.1.4 Starvation Concern:

As explained earlier, if a packet does not encounter enough black bubbles during transit, it passes the remaining count of its marked black bubbles (i.e., *CntH*) to the ejection port. However, for traffic patterns with 1-1 communication, it is very likely that this counter keeps incrementing at a particular ejection point. This means that one router has injected most of the black bubbles in that ring. This condition can lead to starvation of other routers in the ring, which may never inject new black bubbles.

We solved this problem by having a separate backward displacement signal for *CntI*. Every time *CntI* becomes greater than the packet size in terms of worm bubbles, *CntI* is displaced backwards, evening out the black bubbles injected by different routers in the ring (rule 13). Backward displacement of *CntI* means that a router, which has injected too many black bubbles in the ring, is giving those bubbles to its neighboring routers to decrease their injection delay.

5.2 Adaptive Bubble Sharing

The injection limitation in WBFC limits its performance specially at high loads when most ports are marking most of the bubbles as critical, leaving only gray bubbles to allow injection. In such a scheme, allowing support for adaptive routing will increase throughput. However, the problem with adaptive routing is that it requires additional virtual channels and thus buffer space. We present adaptive bubble sharing which merges the ideas presented in bubble-coloring scheme (BCS) [113] with WBFC to provide support for adaptivity in single-VC centralized buffer routers. We next give a brief description of the bubble-coloring scheme.

5.2.1 Bubble-Coloring Scheme

[113] presents bubble coloring scheme (BCS), a method to perform adaptive routing using bubble flow control in packet based networks. The basic idea is to maintain a virtual ring with a critical bubble that connects all the routers of the network, which can be used as an escape path for adaptive routing.

Consider the example of a mesh shown in Figure 35. The dotted line represents a fully connected virtual ring utilizing some channels of the network. A critical bubble is maintained in the ring using the injection/ejection rules of CBS. This bubble will allow

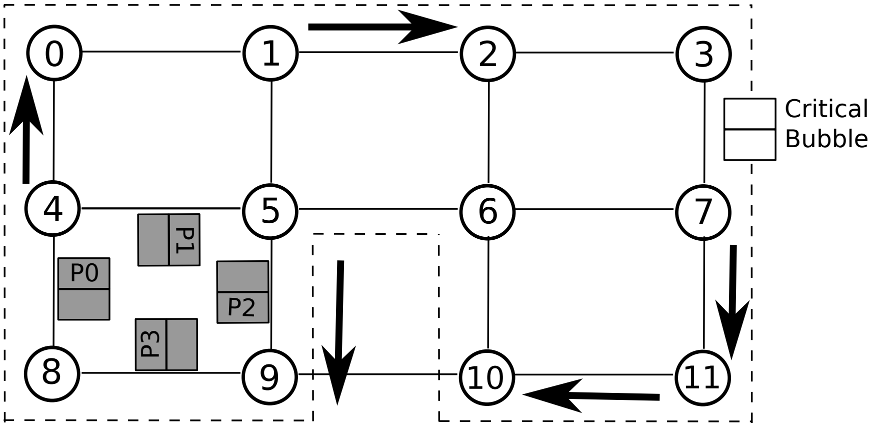


Figure 35: Bubble-coloring scheme (BCS)

packets in the ring to always make forward progress. Packets that are not in the ring can always contest for injection into the ring. e.g. in Fig 35, four packets are waiting on each other in a cycle. However, packet 0 is also contesting for the north port, which is part of the virtual ring. The critical bubble present in the ring will move backwards, allowing packet 0 to escape the cycle. The scheme, however, does not work for wormhole networks, since it does not guarantee that packets ejecting the ring will be drained out completely (discussed later). We have used the basic idea of providing an escape path using a virtual ring to design our adaptive bubble sharing scheme for wormhole based centralized buffer routers.

Next, we present the necessary deadlock avoidance requirements for any adaptive routing scheme to work, which requires three things to be guaranteed:

1. There must always be a deadlock-free escape path from any source to any destination.

2. Packets ejecting the escape path must be consumed.

3. All packets are guaranteed to contest for injection into the escape path.

5.2.2 Satisfying Condition 1

The first condition is satisfied by having a virtual ring with guaranteed bubbles, as used in BCS. For networks with centralized buffers, bubble sharing can be used instead of a packet-based critical bubble used in the original bubble coloring scheme. A problem, however, is that the virtual ring in BCS allows the use of 180 degree turns. With wormhole networks, this can lead to deadlocks within a packet, that is, a packet going towards its minimal direction takes a 180 degree turn to enter the escape ring, and then takes another 180 degree turn towards its minimal direction, deadlocking itself. We avoided this by having two separate virtual rings going in opposite directions, and prohibit 180 degree turns. Since both escape paths are deadlock free, prohibiting one does not break deadlock avoidance guarantees provided by the other.

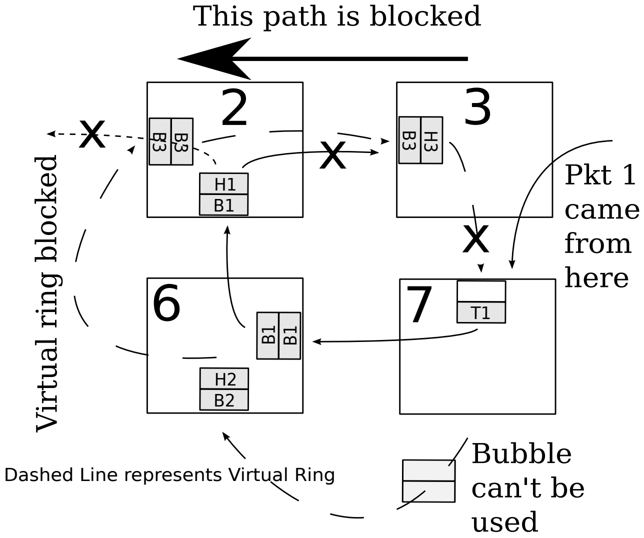


Figure 36: An example showing deadlock with BCS in wormhole networks

5.2.3 Satisfying Condition 2

The second condition is similar to what is generally called the consumption assumption. However, since packets can leave the virtual ring without reaching their destinations, it is possible that the head flit leaves the ring and gets blocked in the non-ring channels of the network, leaving body and tail flits in the virtual ring. Consider the example depicted by Fig 36 in which router 2,3,6 & 7 of a 4x4 network are zoomed in. Assume that the virtual ring is the same as shown earlier in Fig 35. Further, assume that the path in router 2 & 3 going west is blocked. A new packet P1 with destination 1 arrives at router 3 from the east port. Since, the path forward is blocked, it takes the escape virtual ring going south. Suppose at router 7, the minimal direction, i.e. West, is free. So it leaves the ring towards router 6. Router 6 again sends the packet north towards router 2. The scenario is shown in fig 36. In that figure, H1, B1 & T1 represents head, body & tail flit of packet P1 respectively. Suppose another packet P3 is also moving through the routers as shown in the figure. Since, the west direction is blocked, H1 cannot take it. The virtual ring is already occupied by P3. H3 cannot move forward because T1 is waiting at the input buffer of router 7. The bubbles present somewhere else in the virtual ring cannot be propagated to router 7 to break the deadlock. Note that conditions 1 and 3 are satisfied, that is, there are bubbles in the ring and head flit is contesting for the escape path. However, the second condition of consumption is not satisfied. This scenario does not happen in original BCS because, in packet-based networks, when a packet leaves the ring, it always drains, i.e., there are no body and tail flits to lag behind in the previous routers. With central buffers, we can utilize the above mentioned fact by checking a space of *PktS_WB* in the central buffer before ejecting a packet from the virtual ring, ensuring that it will drain completely. Hence, ejection out of the virtual ring is only allowed if $WhiteBubbleCnt > PktS\_WB - 1$, guaranteeing ejection of the complete packet from the ring, and hence breaking the deadlock condition discussed above.

5.2.4 Satisfying Condition 3

The third condition is satisfied in edge buffer routers by allowing head flits to leave only when the input buffers of the downstream router is empty *(credit=input_buffer_size)*. This cannot be used with centralized buffer routers due to the presence of EB links with no credit based flow control. EB links do not guarantee head flits to be at the top of the input buffer contesting for allocation all the time. This means that it is possible that all head flits wait behind the tail flits in a cycle, and are not even allowed to contest for the escape path.

The solution requires a guarantee that once a head flit traveling outside the virtual ring leaves the allocation stage, it will reach the head of the downstream input buffer and contest for the escape path (if required). A simple way to guarantee this is to allow movement only when there is a space of one packet left among the white slots in the central buffer. This will ensure that the current packet drains completely and the subsequent packet's head reaches the top of the input buffer. However, it will put a significant injection bottleneck in the non-ring channels, especially because the virtual ring will be eager to occupy any available white bubbles. We reserve a pool of bubbles, (let's call them yellow bubbles), specifically for the non-ring channels, and prohibit the virtual ring to take these bubbles. Channels not in the ring are allowed to occupy from the pool of yellow and white bubbles while channels within the ring can only take white or their own black, blue, or gray bubbles. Injection in the non-ring channels is allowed as long as $WhiteBubbleCnt + YellowBubbleCnt > PktS\_WB - 1$. This condition is enough for drainage of packets ejecting the ring as well (condition 2), since they are also injecting into the non-ring channels. The introduction of yellow bubbles will allow packets to take minimal non-ring channels more often with a high impact on low load latency. We further prioritize minimal channels over non-minimal channels during allocation reducing low-load latency even further. The extended algorithm for adaptive bubble sharing flow control is given by algorithm 2.

```
Init:
Make 2 opposite fully connected virtual rings;
Initialize according to bubble sharing;
Route Computation:
Provide minimal adaptive channels;
Provide 2 non-minimal escape channels;
Prohibit 180 degree turns;
Injection & Transit:
Same as bubble sharing for both rings;
Ejection:
ejecFlag == ($WhiteBubbleCnt + YellowBubbleCnt > PktS\_WB - 1$);
for All non-ring channels do
    if $ejecFlag == true$ then
        Same as bubble sharing;
    end
end
```

Algorithm 2: Adaptive Bubble Sharing Flow Control

5.3 Modifications to Centralized Buffer Router

The modifications to CBRs required to implement both deterministic and adaptive bubble sharing is given as shaded regions in Fig 37. As explained earlier, the DAMQ-style central buffer is organized as small worm-bubble sized slots. A *color* and a *portno* field is added to each slot to determine the color and the ring assigned to the slot. A logic of few gates is added during the allocation and deallocation of each slot, to determine whether a slot can be assigned a specific color and ring, based on its current color and status. IBSA stage is modified to implement algorithm 1, and rules of adaptive bubble sharing. This, however, works in parallel to the allocation logic, having minimal impact in its critical path. Other modifications like CntI for each ring, CntH in each head flit, progressive movement, and backward displacement signals are added similar to the original WBFC and BCS schemes. Backward displacement hardware is slightly modified to accommodate exchange of CntI signals. Overall, the modifications added a few gates, flip flops, and control signals, with almost negligible impact on the critical path of the router.

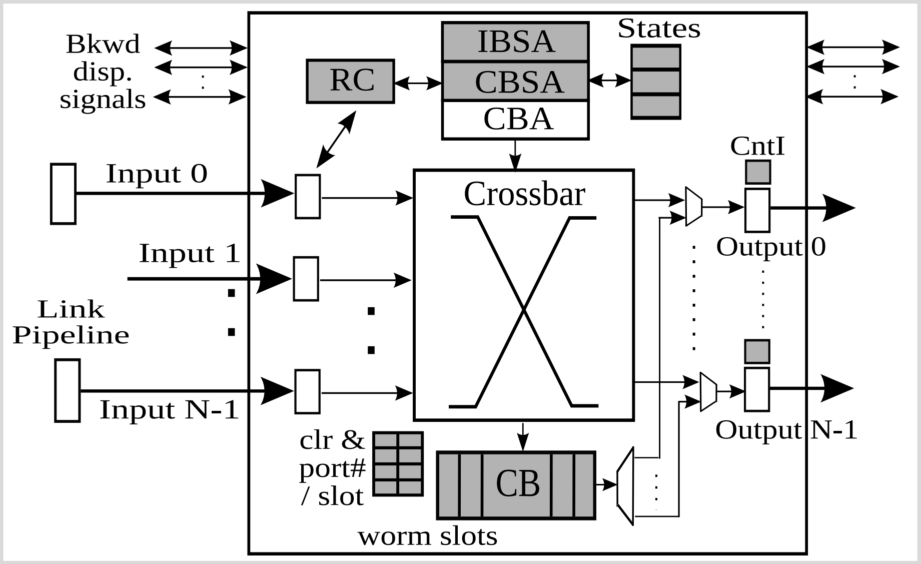


Figure 37: Modified centralized buffer router [32] to support bubble and adaptive bubble sharing

5.3.1 Worm Bubble Coloring (WBCS)

The idea of adaptive routing with single-VC wormhole networks can also be applied to edge buffer routers. We only need to satisfy the three conditions given above. Condition 1 and 3 can be satisfied by WBFC, and by allowing head flits to leave only when *credit=input_buffer_size*. The problem of drainage from the virtual ring can be solved by providing a small *packet_size - worm_bubble_size* entry central buffer. Ejection from the virtual ring is only allowed when this buffer is empty, and there is no other packet leaving any of the virtual ring. If a head flit is already ejected from the virtual ring, a *cntStk* counter is started to count the number of cycles body and tail flits will wait in the router before being allowed to get ejected from the ring. When the counter reaches a threshold (considered a sufficient condition for that packet being blocked), body and tail flits are moved to the central buffer, allowing the head flit of the next packet to enter the input buffer and contest for the virtual ring. We call this scheme as Worm-Bubble-Coloring and used it for evaluation purposes.

5.4 Results

Next we discuss the results of our bubble sharing and adaptive bubble sharing schemes.

5.4.1 Simulation Setup

The proposed schemes are evaluated using the same in-house router micro-architectural simulator that was used with the original CBR design. We also developed three edge buffer router models for comparison: First using Worm Bubble Coloring, second using WBFC, and third using standard Duato's protocol [22] to avoid deadlocks. Any extra VCs in all edge buffer routers use minimal adaptive routing. We used worm size of two flits for central buffers in all our simulations. For adaptive bubble-shared routers, the routing logic provides minimal adaptive paths along with the non-minimal escape rings. We prioritize minimal paths over non-minimal paths during allocation. We assume that the backward displacement or the progressive movement of bubbles can take place in a single cycle regardless of link delays. This can be done by having a separate network for control signals.

Table 7: Buffering configurations of various routers

Router	Abbreviation	InBuff	CBuff
Baseline	Base_VCx	2*x	0
Worm Bubble Flow Control	WBFC_VCx	2*x	0
Adaptivity with Edge Buffers	Worm_BCS_VCx	2*x	4
Bubble Sharing	Bubble_Share_Cx	1	x+8
Adaptive Bubble Sharing	Adp_Bubble_Cx_y	1	x+y+4

Parametric configurations of each of the routers is given in Table 7. The table also shows the abbreviations used in the results section for each of the router. Here *_VCx* means edge buffer router with *x* number of VCs. Hence WBFC_VC2 means a router with worm-bubble flow control and two VCs. Similarly, *_Cx_y* represents bubble sharing routers with *x* entries reserved for white bubbles, and *y* entries reserved for yellow bubbles. Note that the reserved slots for blue bubbles (one per ring) are additional from the slots given by *_Cx_y*. We have assumed one message class and single flit output buffers. We have evaluated our schemes using 2D torus topology with 4x4 & 8x8 networks. We further developed 2D generalized hypercube topology (GHC), which has high number of ports, to understand the impact on high radix routers. Rest of the network parameters are similar to section 4.5.

5.4.2 Performance with Synthetic Traffic

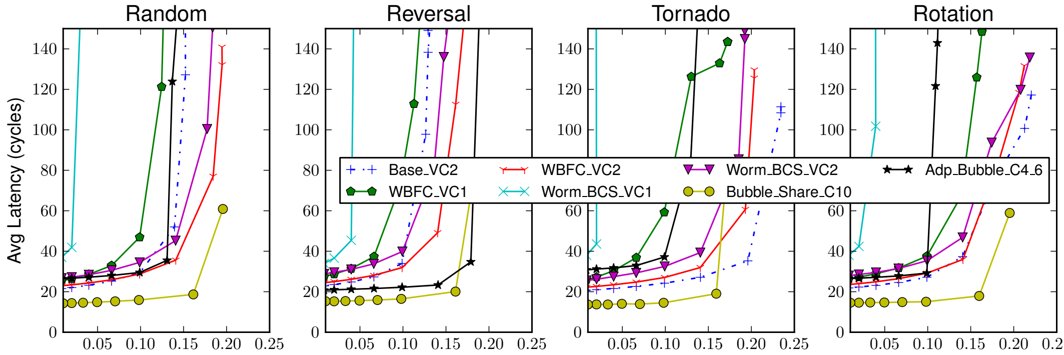


Figure 38: Delivered throughput (retired flits / node / cycle) vs average latency (cycles) for 4x4 torus with different traffic patterns

2D Torus Topologies: Figure 38 compares throughput and average packet latency of various schemes in a 4x4 torus network. The performance of single-VC WBFC and single-VC Worm-BCS is low. This is because of the injection limitations required to enter a ring. However, with 2VCs, the performance increases significantly, more than the baseline 2VC router for most traffic patterns. The reason lies in the availability of more non-ring channels that does not suffer the injection limitation. Bubble sharing gives the best result both in terms of low load latency and saturation throughput. Reduction in low-load latency is attributed to elastic links and single cycle pipeline delay in the router. Furthermore, having multiple bubbles per ring reduces the injection delay, which results in improved throughput as compared to WBFC. Adaptive bubble sharing suffers performance loss, both in terms of no-load latency and saturation throughput, because of the ejection limitations discussed in section 5.2. Since, there are a few number of non-ring channels, ejection limitation does not allow packets that have already entered the ring to leave it, limiting the throughput equal to the throughput of the virtual ring. However, the minimum number of buffers required is extremely low, even with respect to bubble-shared router. This is explained in section 5.4.3.

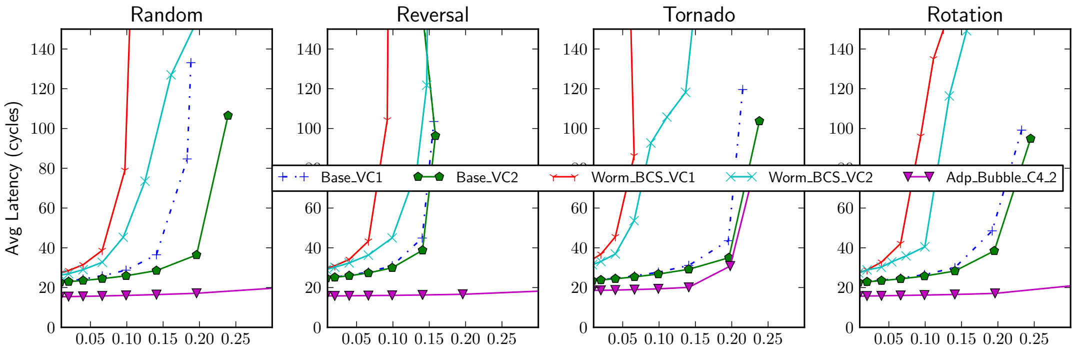


Figure 39: Delivered throughput (retired flits / node / cycle) vs average latency (cycles) for 4x4 GHC

2D GHC Topologies: Figure 39 shows the same result with a 4x4 GHC. In this case, the ejection limitation is reduced because of the high number of non-ring channels available. Furthermore, prioritizing minimal hops reduces the low-load latency of the adaptive bubble sharing router, as well as its saturation throughput. Both edge buffer routers suffer due to the presence of credit base flow control. Figure 40 compares throughput and average packet latency with an 8x8 GHC. It can be seen that the adaptive bubble sharing scheme surpasses others by a large margin. Again, this is because of the high number of non-ring channels available along with EB links. Furthermore, the buffering requirement per router did not change, even with high number of ports. Extremely low no-load latency is also achieved due to the presence of EB links, priority for minimal hops, and low hop count.

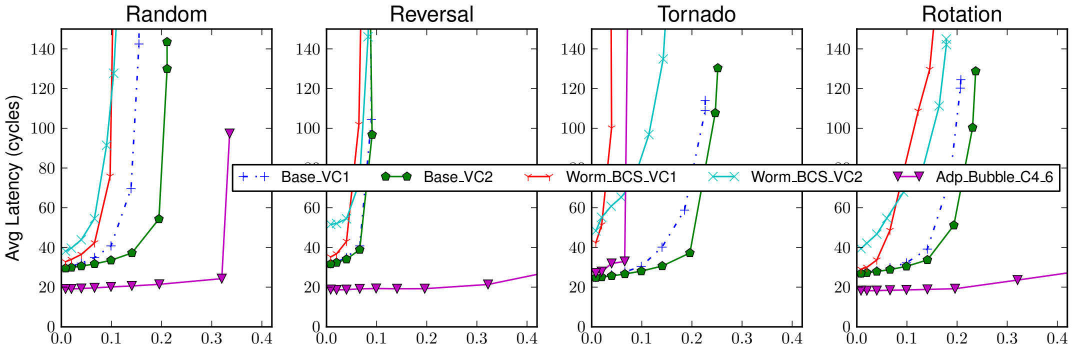


Figure 40: Delivered throughput (retired flits / node / cycle) vs average latency (cycles) for 8x8 GHC

We conclude this section by making the following remarks. For low radix networks, adaptivity using bubbles does not help. In such a case, deterministic bubble sharing scheme performs the best with reduced buffer space due to sharing. However, with high radix topologies that have many routes to destination, adaptivity provided with bubbles significantly improves performance without increasing the buffer area.

5.4.3 Buffer Space Analysis

Edge buffer routers requires at least one worm-sized entry per input port per VC. Worm_BCS also requires a *packet_size* - *worm_size* central buffer for minimum operation. WBFC requires *PktS_WB black and one gray* bubble to be injected per ring at initialization. For a six flit packet with *bubble_size* of two, this means four bubbles per ring. The total number of rings in a 4x4 torus are 16, four in each direction. Thus, 4x16=64 bubbles will be injected at initialization. Edge buffer routers use input buffers to provide these bubbles. With central buffers, minimum number of entries required by each router is 64/16=4 bubbles. Assuming that we have at least one white bubble per router at initialization, the number of entries required at each router = 5 worm-bubbles or 10 flits. Furthermore, each router requires one blue bubble per ring making the minimum central buffer size to be 18 entries for routers with bubble sharing. It should be noted that with an 8x8 torus, this will be reduced to 12 entries per router. In the adaptive bubble sharing case, the two virtual rings require 4*2=8 bubbles anywhere in the network, in addition to the two blue bubbles per router. This makes the minimum requirement with adaptive bubble sharing to be 6-8 entries per router.

Table 8: Buffer space per router (bytes) with different configurations

RowNo	**Parameter**	**2D-Torus**	**4x4-GHC**	**8x8-GHC**
1	**Baseline_VC2**	400	560	1200
2	**WBFC_VC2**	400	560	1200
3	**Worm_BCS_VC2**	464	624	1264
4	**Bubble_Share_C10**	448	512	768
5	**Bubble_Share_C12**	480	544	800
6	**Adp_Bubble_C4_2**	320	384	640
7	**Adp_Bubble_C4_6**	384	480	704

Table 8 gives the total buffer space requirements of different routers with different configurations. The formula for calculating the buffer space is $[P*(I*VC+O)+C]*L$, where L is the link width, P is the number of ports, I, O, C is number of flits in the input, output and central buffers, respectively. For 2D-ring topologies, the 2VC baseline and WBFC routers require 400 bytes per router. Worm_BCS requires slightly more with a small central buffer. Since, *worm_size* is only two flits, and we have one flit staging buffer in the centralized buffer routers, the total buffer space in bubble sharing router with central buffer entries of 20 is high. However, the buffer size increases very slowly with increase in the number of ports and gets extremely low, e.g., for 8x8 GHC. The adaptive bubble sharing router will always have smaller buffering requirement than others because of its small entry central buffer. Furthermore, its throughput with large number of adaptive channels is high making it an ideal candidate for high-radix routers.

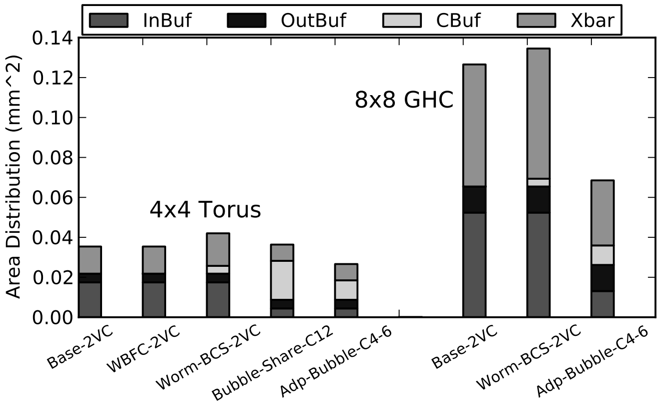


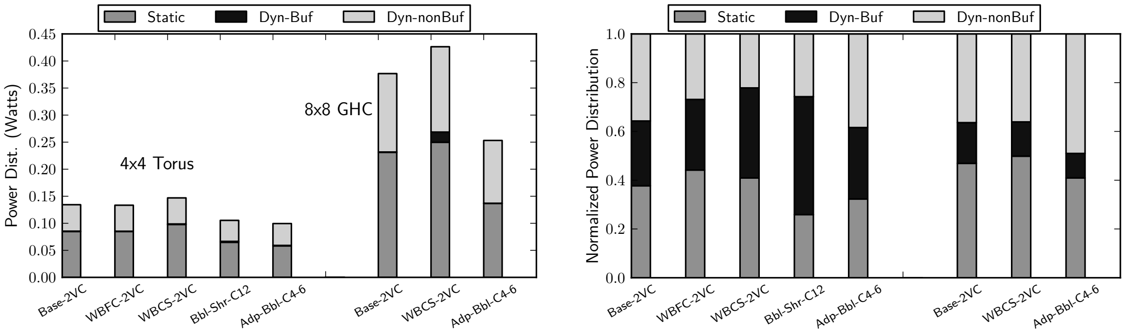
Figure 41: Router area distribution with different configurations

5.4.4 Impact on Area

Fig 41 gives the area distribution of a 4x4 Torus and an 8x8 GHC router with different configurations. Baseline has a large area in the input buffer for both 4x4 and 8x8 routers even with a single message class. The input buffer area for bubble sharing routers is low. However, central buffer takes a significant amount of area as well. In 2D-Torus, this area dominates over other parts making bubble shared routers 3% more expensive than the baseline 2VC router. However, with adaptive bubble sharing and small buffers, the area decreases by 27% making it the cheapest. Furthermore, with high radix, such as in GHC, the increase in central buffer area is very low, as compared to area requirements of multi-VC input buffers and crossbars, thus reducing the area of adaptive bubble router by 46% and 52% compared to the Base_VC2 and Worm-BCS_VC2 routers, respectively. Note that the crossbars are configured as *input_channels* X *output_ports*, that makes their area larger depending upon the number of VCs. Since centralized buffer routers do not have VCs, their crossbar area does not increase as significantly as edge buffer routers.

5.4.5 Impact on Power

Figure 42a compares the static and dynamic power dissipation of different routers at low loads configured in a 4x4 torus and 8x8 GHC topology. At low loads, most of the router power is static, with very small dynamic power in the buffers. The static power of the



(a) Low Load

(b) High Load

Figure 42: Router power distribution with different configurations: a) Low-load power distribution, b) high-load power distribution

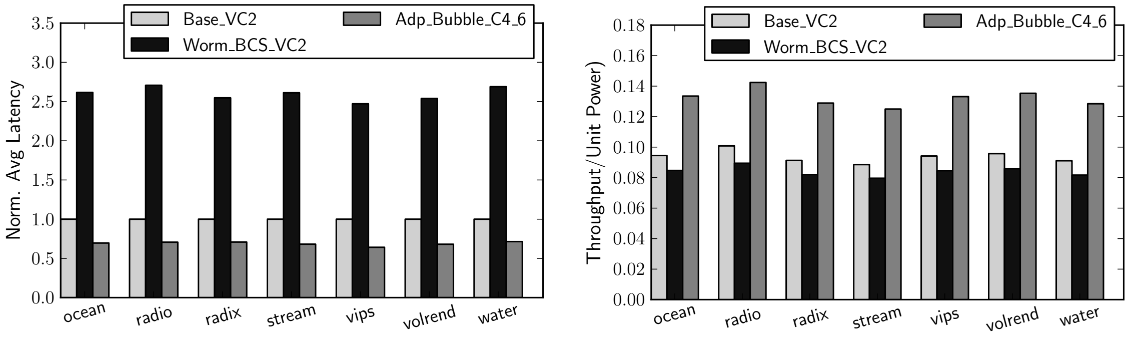
bubble shared router is 24% lower than the baseline in the 4x4 torus topology. This is due to the presence of a smaller crossbar (single-VC) in it. Adaptive bubble shared routers further reduces it to 32% & 41% for 4x4 torus and 8x8 GHC, respectively, because of their smaller central buffers.

At high loads, as shown in Figure 42b, buffers take a significant portion of the overall dynamic power, with reduced distribution in links & crossbars. The highest for torus topology is taken by bubble sharing routers because of the largest buffer size. However, the adaptive bubble sharing case, as can be seen with the GHC topology, requires the least amount of dynamic buffer power even with the highest throughput.

We conclude this section by pointing out the fact that bubble sharing not only reduces the buffer space, it also removes the need of virtual channels. This may lead to reduction in crossbar area as well along with reduced buffering.

5.4.6 Results with Real Benchmarks

Fig 43a gives the average packet latency of various schemes with an 8x8 GHC topology normalized with respect to the baseline 2VC router. As can be seen, adaptive bubble sharing consistently gives lower latency across a range of benchmarks. The percentage improvement on average is 31%. Similar results can be seen in Fig 43b showing throughput per unit power. Average percentage improvement in this case is 41%. With 8x8 torus (Fig 44),



(a) Avg Packet Latency (8x8 GHC)

(b) Throughput Per Unit Power (8x8 GHC)

Figure 43: Results with benchmark traces - Normalized average packet latency and throughput per unit power for 8x8 GHC

although adaptive bubble sharing did not perform as good, our bubble sharing scheme outperforms all other schemes with an average improvement in throughput per unit power by 13% and 25% compared to Base_VC2 and WBFC_VC2, respectively.

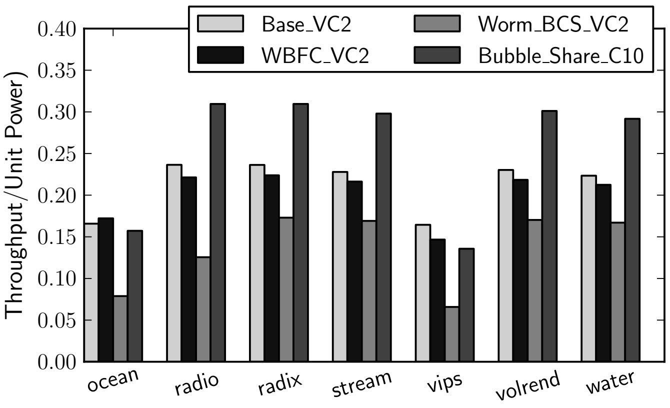


Figure 44: Benchmark Traces Results - Throughput per unit power for 8x8 torus

5.5 Concluding Remarks

This chapter addressed the buffer space reduction problem in centralized buffer routers by proposing the use of variants of bubble flow control; specially for high-radix networks. A wormhole-based version of the bubble coloring scheme is also presented to provide adaptivity without the use of VCs in CBRs. However, the design only works for regular topologies like torus with fixed cyclic paths or rings. Bubble flow control and its variants may not be the ideal choice for more complex, irregular topologies with non-deterministic cycles in them. The next chapter extends CBRs to support multiple VCs and use them with an irregular high-radix topology reducing its buffering requirement while maintaining deadlock freedom with the help of multiple virtual channels.

CHAPTER 6

DETERMINISTIC CBR WITH VCS FOR IRREGULAR TOPOLOGIES

Centralized buffer routers uses elastic buffer (EB) links to reduce the latency penalty of long wires. EB links, however, does not allow the use of virtual channels in the links. Hence, we used variations of bubble-flow control to provide deadlock freedom. However, bubble-flow control only works for regular topologies, such as rings and torus. For complex irregular topologies, we extend the centralized buffer design to support multiple VCs. We used recently proposed ElastiStore [84] technique to allow virtual channel with CBRs. To explore VC-based CBRs with high-radix irregular topologies, we used the VC-enabled CBR design with Slim Fly, a recently proposed high-radix topology by Besta and Hoefler [5], to come up with an optimum diameter-2 on-chip network with the least number of ports in the router. The topology is fairly irregular and requires two VCs for deadlock freedom, hence CBRs with multi-VC support are used.

6.1 Multi-VC Centralized Buffer Router

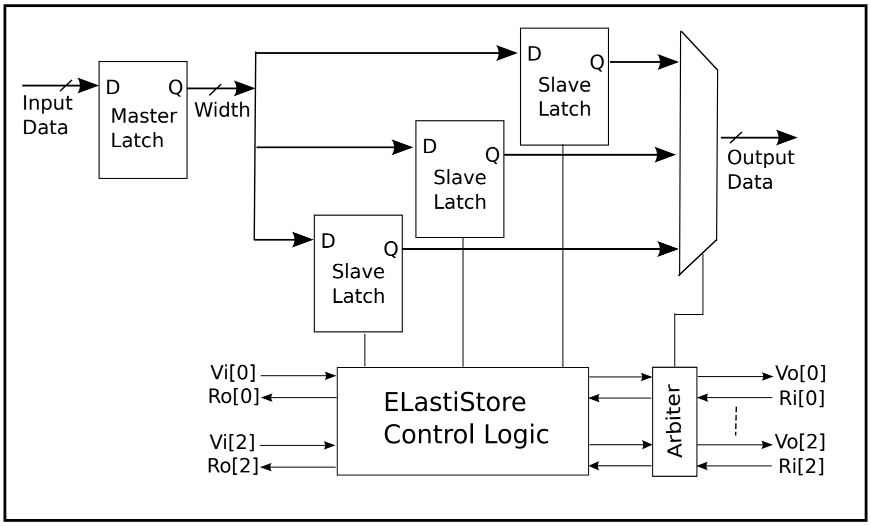


Figure 45: A basic ElastiStore link

The main modification in a multi-VC CBR is the use of ElastiStore (ES) links instead of elastic-buffer (EB) links. ES links, as shown in Figure 45, use a separate pipeline buffer and the associated control logic for each VC in the links along with a per-VC ready-valid handshake signal. The per-VC ready-valid handshake signals independently progress flits of each VC, removing their mutual dependence in the pipelined link. The design is further extended by only keeping the slave latch per VC and sharing the master latch among all VCs. This reduces the overall area and power due to ES links. The performance cost is minimal and reaches $\frac{1}{|VC|}$ only when all VCs except one are blocked in the pipeline. Other modifications include using per-VC (instead of per-port) I/O staging buffers and head/tail pointers of the CB, ensuring mutual VC independence. The crossbar radix is still $k' \cdot (k'+1)$, like in the original CBR. This is enabled by having small multiplexer / demultiplexer before and after the crossbar inputs and outputs. We maintain single input and single output for the CB, which does not impact performance significantly [4].

To explore VC-based CBRs with high-radix irregular topologies, we used the VC-enabled CBR design with Slim Fly, a recently proposed high-radix topology by Besta and Hoefler [5], to come up with an optimum diameter-2 on-chip network with the least number of ports in the router. Smaller diameter and thus hop count meant lower latency while fewer number of ports resulted in lower power. The design, however, has long multi-cycle links which can result into large buffers at the inputs and severe performance loss due to credit-based flow control. Central buffers provide an ideal solution in this case with their single-flit input buffers and elastic-buffered links. Furthermore, the topology is fairly irregular and requires two VCs for deadlock freedom, hence CBRs with multi-VC support is used. We next present a brief description of rack-based Slim Fly Topology and its extension to on-chip networks.

6.2 The Slim Fly Topology (SF)

SF [5] is a low-diameter cost effective topology for large computing centers that uses mathematical optimizations to minimize the network diameter D for a given radix k while maintaining full global bandwidth. The main SF design uses graphs introduced by McKay, Miller, and Širán [57] (denoted as *MMS graphs*) to connect routers and it has a diameter of 2 ensuring lowest latency for many traffic patterns [5]. More importantly, decreasing D also reduces the number of required network resources (packets traverse fewer routers and cables), directly translating to reductions in cost and power consumption.

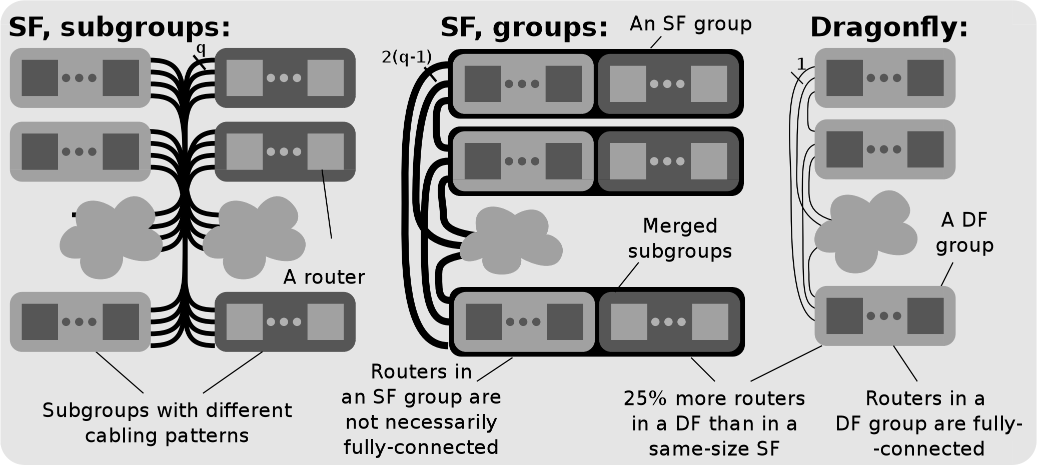

SF, subgroups:
q
A router
Subgroups with different cabling patterns
SF, groups:
An SF group
2(q-1)
Merged subgroups
Routers in an SF group are not necessarily fully-connected
25% more routers in a DF than in a same-size SF
Dragonfly:
1
A DF group
Routers in a DF group are fully--connected

Figure 46: High-level overview of Slim Fly topology

SF has a highly symmetric structure presented in Figure 46. Here, we illustrate the intuition behind the SF structure necessary for the on-chip design; the details are discussed by Besta and Hoefler [5]. Routers in SF are grouped into SF *subgroups* with the same given number of routers (denoted as q). There are two types of subgroups, each type with the same pattern of intra-group cables. Every two subgroups of different types are connected with the same number of cables (also q). There are no cables between subgroups of the same type. Thus, subgroups form a fully-connected *bipartite* graph where each edge is formed by q cables. Subgroups of different types can be merged pair-wise into identical SF *groups*, each with $2q$ routers. Groups form a fully-connected graph where each edge consists of $2(q-1)$ cables. q is a parameter that determines the structure of a specific SF [5].

An SF with a given q is characterized by the following parameters [5]: the number of routers $N_r = 2q^2$, the network radix $k' = \frac{3q-u}{2}$, and the number of endpoints $N = N_r p$. The concentration p is equal to $\lfloor k'/2 \rfloor + \kappa$; κ determines the desired tradeoff between increasing endpoint density (larger κ) and decreasing contention (smaller κ).

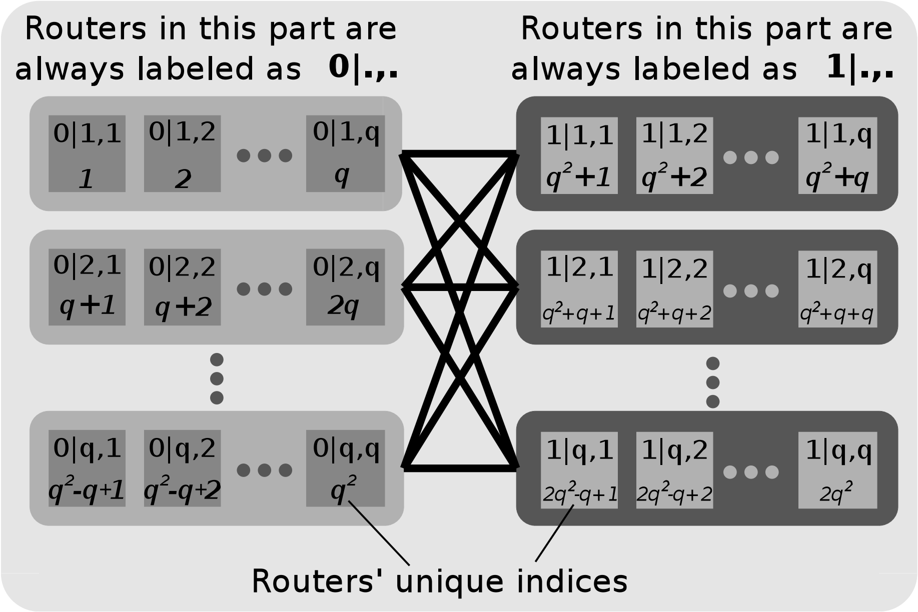


Figure 47: Labeling and respective indexes of SF routers

Each SF router is labeled as $[G|a, b]$. $G \in \{0, 1\}$ determines the type of a subgroup that the router belongs to, a is the number of the subgroup of type 0 or 1, and b is the router's ID in the subgroup; $(a, b) \in \{1, 2, ..., q\}^2$. The index i ($1 \leq i \leq N_r$), assigned to each router can be computed as $i = G \cdot q^2 + (a - 1) \cdot q + b$, for a given label $[G|a, b]$. Figure 47 presents the labeling and respective indexes of different routers.

A model for placing routers on a 2D grid is illustrated in Figure 48. Here, a router i ($1 \leq i \leq N_r$) is assigned to coordinates (x_i, y_i) on a 2D grid. We assume that routers form a rectangle where $1 \leq x_i \leq X$ and $1 \leq y_i \leq Y$. For two connected routers i and j, the connecting link is placed along the line segments determined by the points (x_i, y_i), (x_i, y_j), and (x_j, y_j) (if $|x_i - x_j| > |y_i - y_j|$), or (x_i, y_i), (x_j, y_i), and (x_j, y_j) (if $|x_i - x_j| \leq |y_i - y_j|$).

Intuitively, SF is similar to the *balanced dragonfly* DF topology [44]. DF also consists of groups of routers that form a fully-connected graph. Yet, there is only one cable between

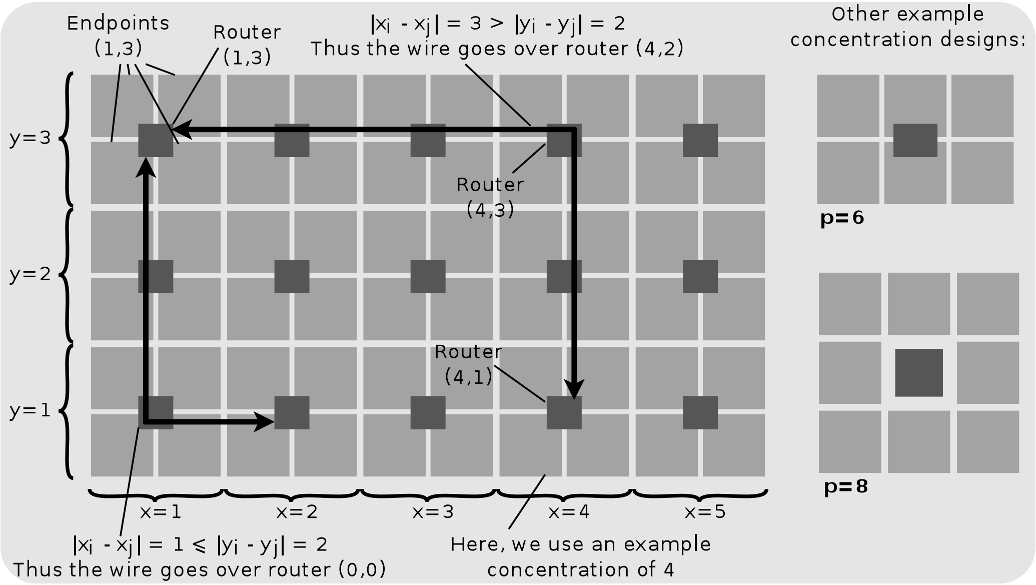


Figure 48: The layout model with two example wires

two groups in DF resulting in higher D. Moreover, the internal structure of each DF group also forms a fully-connected graph, which is not necessarily true for SF. Finally, SF reduces the number of routers by ≈25% in comparison to a DF with comparable N.

6.3 ON-CHIP SLIM FLY

We now describe an on-chip SF design. SF minimizes the radix k, reducing the total buffering size Δ. However, SF's low diameter may need a large number of longer multi-cycle links. In such a scenario bigger buffers would be required to fully utilize the wire bandwidth, overshadowing the advantages provided by small k. To alleviate this, we use two strategies:

Optimizing Layouts To keep the wires short, we propose several physical on-chip SF layouts that lower the average Manhattan distance M between connected routers and reduce edge buffers and latency. This strategy reduces the area/power consumption in on-chip SFs without the need to incorporate any additional micro-architectural mechanisms. Orthogonally, we use SMART links to further reduce wire latencies.

Incorporating Central Buffers We add CBRs to SF and illustrate that this reduces buffering requirement even further. This strategy uses state-of-the-art on-chip schemes that can be manufactured in the foreseeable future.

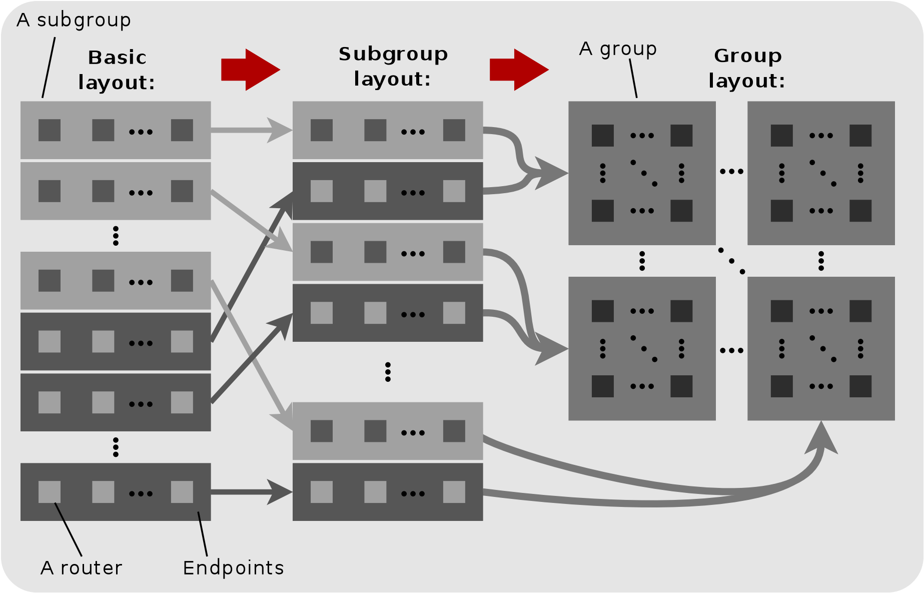


Figure 49: Different physical SF *layouts*

6.3.1 Slim Fly Layouts

We created four generic on-chip SF layouts that minimize average wire length and buffer space as shown in Figure 49. The first one is the *basic layout* (sf_basic) in which subgroups with identical intra-subgroup connections are grouped together and a router $[G|a, b]$ is assigned coordinates $(b, a + Gq)$. As such subgroups are not directly connected, this layout may increase the length of inter-subgroup connections. Thus, we mix subgroups pairwise to reduce the length of wires between subgroups and obtain the *subgroup layout* (sf_subgr). A router $[G|a, b]$ has coordinates $(b, 2a - (1 - G))$ in the subgroup layout. Both sf_basic and sf_subgr have a rectangular shape ($q \times 2q$ routers) and thus can easily be manufactured. Finally, to reduce the wiring complexity, we create the *group layout* (sf_gr) by merging subgroups of different types pairwise and grouping the resulting groups in a shape as close to a square as possible. Intuitively, there are q groups, each group has identical intra-group connections, and there are $2(q - 1)$ wires between every two groups.

6.3.2 Ensuring Deadlock Freedom

SF with $D = 2$ uses two VCs to avoid deadlocks: VC0 for the first and VC1 for the second hop (assuming paths of lengths up to 2). Here, the only dependency is that of VC0 on VC1, which is not enough to form cycles. We now extend this scheme to the CBR design. To ensure deadlock-freedom, two conditions must be met. First, CB's allocation must be atomic: it cannot happen that some flits have entered the CB and the rest is stalled in the links waiting for the CB to get free. This will create a false dependency of one port on the other. Second, head flits of all the packets in different ports and VCs should always be able to compete for the allocation of the output ports and VCs, that is, they must always reach the head of the input buffers and not wait in the link pipelines behind body and tail flits indefinitely.

The first condition is satisfied by reserving a complete packet space during the CB allocation stage. Thus, once a packet decides to take the central buffer path, it is guaranteed to move completely into the central buffer. A packet in the central buffer is always treated as part of the output buffer of the corresponding port and VC. Thus, if the baseline routing algorithm is deadlock free, it will be deadlock free with the central buffers too. The second condition always holds true in the case of edge buffer routers since a head flit is only allowed to move forward when the downstream buffer is completely empty. With no credit based flow control in the EB links, this condition is not guaranteed. We only focus on deadlock-free deterministic routing algorithms. In such a case, head flits are guaranteed to reach the head of the input buffer. But since we are only focusing on deterministic routing with fixed paths, this condition is not required for deadlock-freedom.

This fact, however, indicates that our design does not support adaptive routing with the CBRs. We plan to extend it to support adaptivity in future.

6.4 EVALUATION OF ON-CHIP SLIM FLY

The next section compares SF's using CBRs with other topologies and buffering options.

6.4.1 Simulation Setup

Comparison Targets We compare SF to: concentrated 2D meshes (CM, $D = \lceil \sqrt{N_r} - 2 \rceil$), tori (T2D, $D = \lceil 1/2 \sqrt{N_r} \rceil$), and two-level flattened butterflies [45] (FBF, $D = 2$). CM and T2D represent low-radix baselines. FBF represents a state-of-the-art high-radix topology with $D = 2$. Yet, for a fixed N and for $D = 2$, k is much higher in FBF than in SF. Thus, for fair comparison, we developed a *partitioned* FBF (PFBF) with fewer links to keep k equal to SF's k.

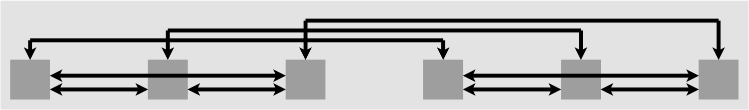


Figure 50: An example partitioned FBF (1D). Cores are not shown.

To get PFBF we partition an original FBF into smaller identical FBFs. The smaller FBFs are connected with one another by one port per node in each dimension. An example is shown in Figure 50. Here, router nodes are divided into two groups of three, each having its own butterfly structure. A single link is connected from a node of first partition to the corresponding node of the second partition. Thus, if a packet has to travel from 0-4, it first takes the inter-partition link to reach 3 and then takes the intra-partition link to reach router 4. It can be seen, that the number of links per dimension has been reduced to 3 instead of 5 as is the case with a full flattened butterfly. The manhattan distance between the two nodes remains the same in most cases, however, the diameter of the 2D network increases to four.

We consider two classes of network sizes: $N \approx 200$ and $N = 1,296$. Network configuration details are presented in Table 9.

Router Architectures We use both routers with edge buffers and with central buffers. An edge router has a standard 2-stage pipeline with two VCs [68]. A CB router takes two cycles in the bypass path and four cycles in the buffered path; we turned off lookahead switch allocation for this analysis.

Buffering Strategies We use: InBuf-F-S and InBuf-F-L (all edge buffers have

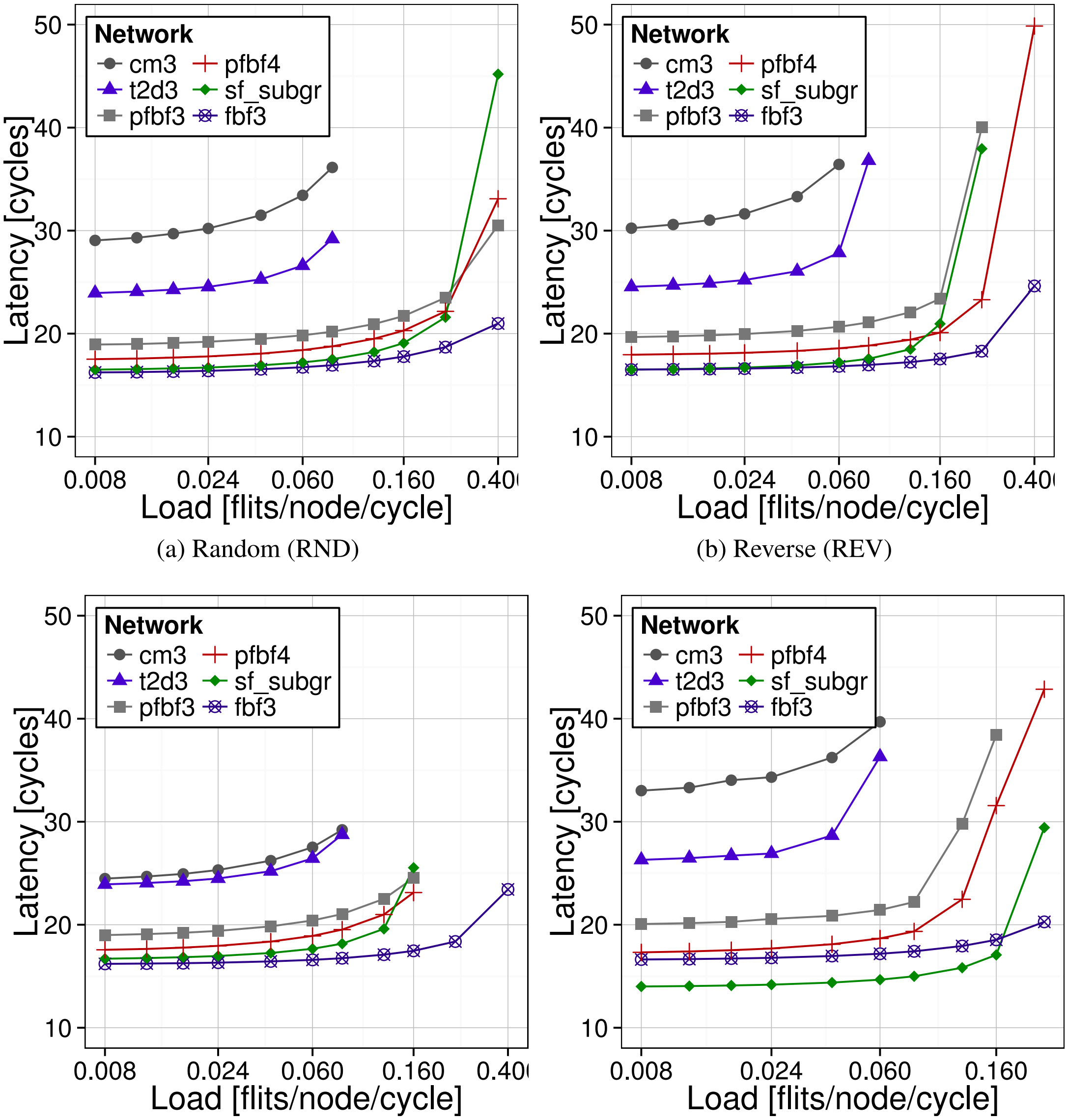
Table 9: Considered network configurations

Network	$N \approx 200$					$N = 1,296$				
	Sym.	p	k'	k	Routers	Sym.	p	k'	k	Routers
T2D	t2d3	3	4	7	8x8	t2d9	9	4	13	12x12
	t2d4	4	4	8	10x5	t2d8	8	4	12	18x9
CM	cm3	3	4	7	8x8	cm9	9	4	13	12x12
	cm4	4	4	8	10x5	cm8	8	4	12	18x9
FBF	fbf3	3	14	17	8x8	fbf9	9	22	31	12x12
	fbf4	4	13	17	10x5	fbf8	8	25	33	18x9
PFBF	pfbf3	3	8	11	4 FBFs (4x4 each)	pfbf9	9	12	21	4 FBFs (6x6 each)
	pfbf4	4	9	13	2 FBFs (5x5 each)	pfbf8	8	17	25	2 FBFs (9x9 each)
SF	sf_*	4	7	11	10x5	sf_*	8	13	21	18x9

the size of 5 and 15 flits, respectively), InBuf-O-S and InBuf-O-N (edge buffers have minimal possible sizes for 100% link utilization with/without SMART links), CBR-S and CBR-L (CBs of sizes 6/40), and Only-EB (neither edge buffers nor CB). Other simulation parameters are similar to the previous two chapters. We also used an adversarial traffic pattern ADV1 which maximizes the load on a single-link path. Simulations with synthetic traffic patterns are performed for 1M cycles only because of the large size of the networks. Multiple copies of 64-threaded version of each benchmark is used to model real benchmark traffic of large networks.

6.4.2 Performance: Synthetic Traffic

We start the analysis by comparing performance of SF to other topologies using synthetic traffic. The analysis results are presented in Figures 51 and 52. As expected, SF outperforms CM and T2D for all the analyzed cases. For example, for RND and $N = 1,296$, it improves latency by ≈45% (over T2D) and ≈57% (over CM), and throughput by 10x. This is a direct consequence of SF's significantly lower D and higher bandwidth. Then, SF's throughput is marginally lower than that of PFBF in some cases (e.g., $N \approx 200$, REV) because of PFBF's minimum Manhattan paths. Yet, in most cases SF has a higher throughput



(a) Random (RND)

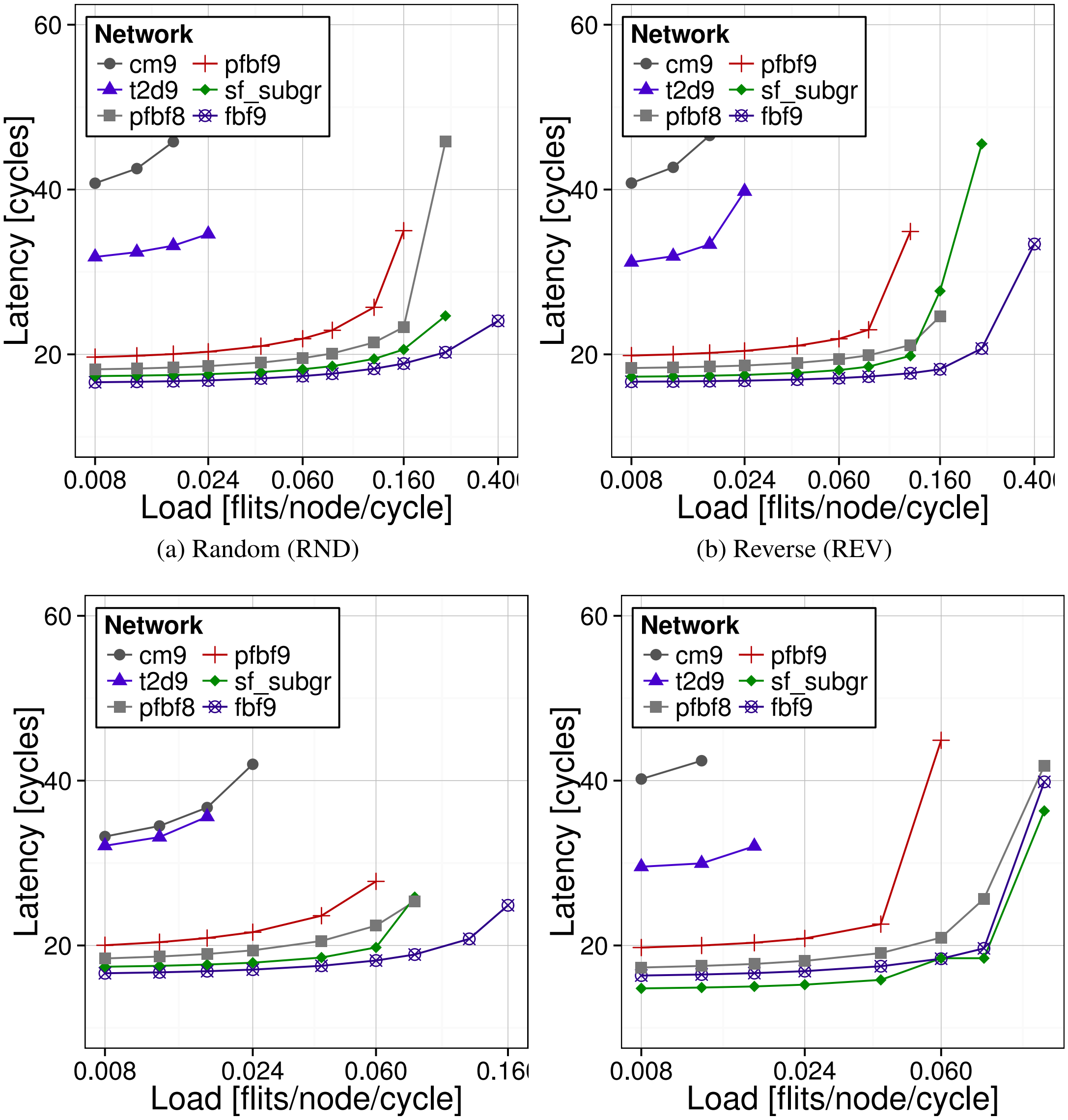
(b) Reverse (REV)

(c) Shuffle (SHF)

(d) Adversarial (ADV1)

Figure 51: Performance analysis (synthetic traffic) of networks with $N \approx 200$ endpoints and with SMART links

(e.g., >60% for $N = 1,296$ and RND). SF's latency is always lower (≈6-25%) than that of PFBF due to its lower D. Finally, without SMART links, SF's longer wires result in higher latency than in FBF (≈%26 for RND and $N = 1,296$). With SMART links, both topologies consistently have comparable latency. Yet, SF delivers lower throughput (≈40%) due to its smaller radix. We will later (section 6.4.4) illustrate that SF offers a lower energy-delay product than that of FBF, achieving a better performance/power tradeoff.



(a) Random (RND)

(b) Reverse (REV)

(c) Shuffle (SHF)

(d) Adversarial (ADV1)

Figure 52: Performance analysis (synthetic traffic) of networks with $N = 1,296$ endpoints and with SMART links

6.4.3 Buffer Sizes

We now present the total buffer size in various networks with different buffering strategies; see Table 10. All the designs with edge buffers require significantly larger buffer space than the CBR designs; the higher the radix is, the larger is the total buffer space required by routers with edge buffers. With central buffers the buffer space does increase with higher radix but the rate of increase is very slow due to single-flit I/O staging buffers. `Only-EB` has

Table 10: Buffering requirements [KB] for different configurations

Strategy	$N \approx 200$					$N \approx 1,296$			
	CM	FBF	PFBF	SF_subgr	SF_gr	CM	FBF	PFBF	SF_subgr
InBuf-F-S	267	477	351	305	305	1424	2275	1802	1926
InBuf-F-L	547	1157	791	649	649	2594	5065	3692	4052
InBuf-O-N	267	701	431	433	459	1424	3594	2222	3513
InBuf-O-S	267	477	351	306	310	1424	2293	1802	2001
CBR-S	161	211	181	173	173	968	1172	1060	1091
CBR-L	195	245	215	199	199	1046	1249	1136	1177
Only-EB	155	205	175	167	167	956	1159	1046	1076

the smallest buffering requirement but it suffers from head-of-line-blocking as mentioned in chapter 4.5. SF gives an optimal tradeoff of k (hence the crossbar size) for a given diameter and thus it has the lowest buffering space with a diameter-2 topology.

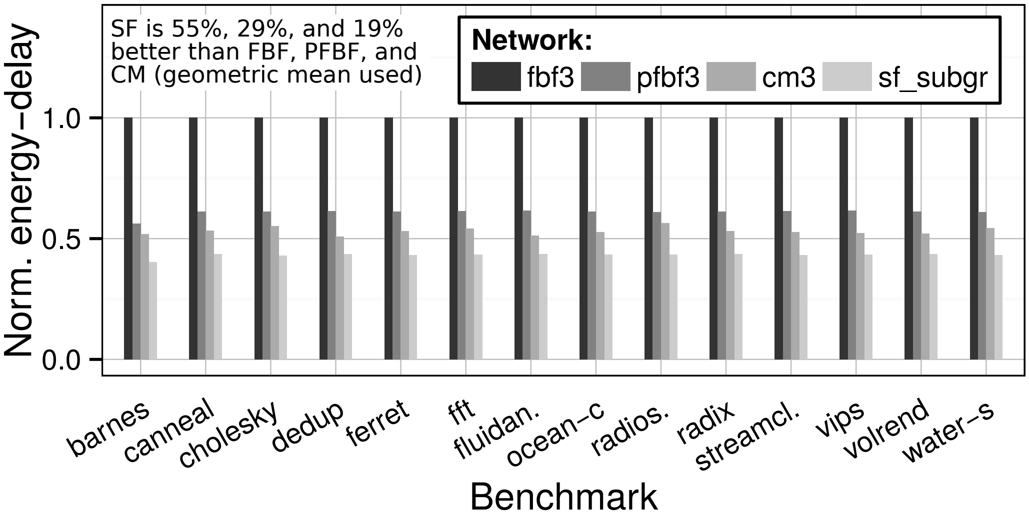


Figure 53: (section 6.4.4) PARSEC/SPLASH benchmarks (with SMART).

6.4.4 Performance & Power: Traces

We now compare SF to other topologies by running traces of PARSEC/SPLASH benchmarks; see Figure 53. As a summary, we illustrate energy-delay (ED) product for each benchmark. SF reduces ED by 55% on average (geometric mean) compared to FBF as it consumes less static and dynamic power. Moreover, SF's ED product is 29% smaller than that of PFBF due to the latter's higher latencies and power consumption. Similarly, SF reduces the ED product by 19% compared to CM. After normalizing the results per router, SF retains its advantages over FBFs, with its ED product being on average 45% and 10% lower than in fbf3 and pfbf3.

6.5 Concluding Remarks

The previous three chapters proposed centralized buffer routers that attempts to reduce latency in on-chip wormhole networks while keeping the buffering space and thus power of the routers small. We explored both deterministic and adaptive versions of CBR and analyzed their performance and power with both regular and irregular topologies under different traffic patterns. Such low-cost, low-latency networks are ideally suited for a 3D stacked system with large number of memory channels. As mentioned before, network latency, address management, and thermal regulation are the three major challenges of 3D stacked memories. Until now, we have explored memory re-organization, network traffic and latency reduction, and address management to improve performance of 3D memories. Next, we shift our focus to thermal problems of 3D stacks. More specifically, we analyze the rise in operating temperature for 3D stacks and propose optimizations to reduce the negative impact of high temperature on the performance of the overall system.

CHAPTER 7

THERMAL ANALYSIS OF 3D MEMORIES

A fundamental problem with 3D ICs is the removal of heat from multiple dies stacked on top of each other leading to various new thermal challenges. These include, 1) higher thermal coupling among processor and memory dies, 2) higher variation in heat removal capability, power density, and temperature; thus rapid thermal degradation among various dies (logic vs. DRAM), and channels within the die (uneven usage of parallel channels), and 3) lower charge retaining capability of DRAMs because of higher temperature; thus more refreshes and reduced DRAM bandwidth availability. These problems can make thermal constraints a major bottleneck in a 3D stacked memory system. Increased temperature also has a negative impact on lifetime reliability of the memory system. Thus, 3D memories have mainly be used with memory-intensive applications running at low frequencies which inherently have low performance. Further improvement in performance scaling by increasing core complexity or frequency is generally limited by thermal constraints of 3D memory. We identify 3D memory's thermal wall problem and explore the performance scaling improvements made possible by using microfluidics cooling technology in a 3D stacked memory system. We also analyzes and compares the reliability-performance tradeoffs between multicore systems with 3D and 2.5D stacked memory identifying correlations between frequency, performance, temperature, and reliability and uses it to quantify the reliability-performance tradeoff.

7.1 Evaluation Methodology

Figure 54 depicts simulation flow used for our experiments. We used Manifold microarchitecture simulator [112] to compose 3D network and memory controllers (MC) connecting 32 out-of-order cores that execute PARSEC parallel application threads [10] as described in 3.2. We compared our 3D system configuration with a comparable silicon

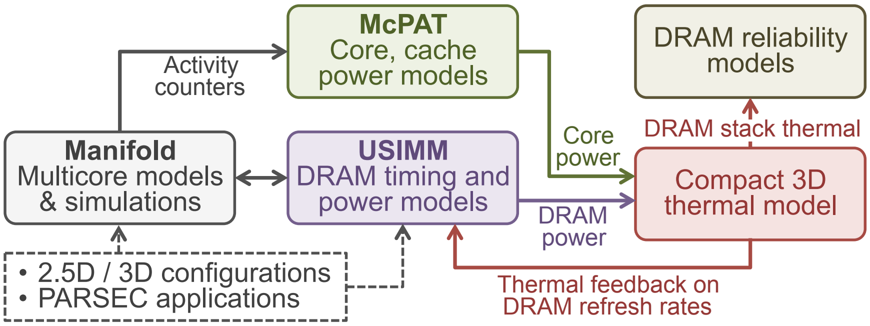


Figure 54: Evaluation methodology for thermal analysis of 3D stacked memories

interposer based memory system. We called it 2.5D a stacked memory system as shown in Figure 55.

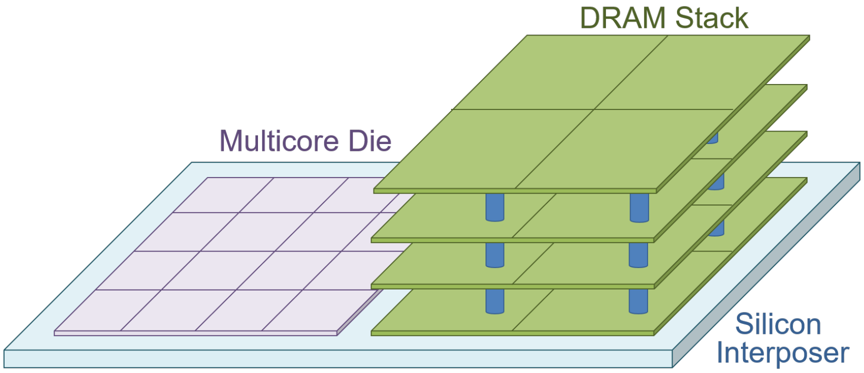


Figure 55: 2.5D Stacked Memory System for Comparison Purposes

For 2.5D stack, the base logic layer consists of four MCs connected through four parallel memory channels. The four MCs each control a quarter of a die, with multiple dies sharing a channel similar to the 3D system. The bank size of both DRAM systems are kept constant to 64MB with the same number of rows and page sizes. However, the 2.5D system has eight banks per rank (due to lower number of channels) as compared to two banks in the 3D stack. This changes the refresh time t_{RFC} for both memories which is 160ns for 3D and 300ns for 2.5D-stacked DRAM. This also impacts various current values which are given in Table 11. Other timing parameters remain the same for both DRAMs; see Table 11. The total DRAM capacity in both the systems is set as 8GB.

Architectural activity counters calculated from Manifold simulations are supplied to

Table 11: DRAM parameters [60] [95]

Parameter	Value-both 3D and 2.5D	Parameter	2.5D-value	3D-value
t_{CLK}	1.25 ns	$IDD0$	95	85
t_{RP}	11	$IDD2P$	3	2.6
t_{RCD}	11	$IDD2N$	32	8
t_{RC}	39	$IDD3P$	19	5
t_{RAS}	28	$IDD3N$	47	12
t_{CAS}	11	$IDD4R$	265	66
t_{WR}	12	$IDD4W$	294	73
t_{RRD}:	5	$IDD5$	200	50
t_{RTP}	6	$IDD6$	4.3	1.1
t_{WTR}	6	t_{RFC}	300ns	160ns

Kitfox [90] which uses McPAT [49] to estimate core area and power. We adopted USIMM [14] to simulate 3D and 2.5D stacked DRAMs and estimate the power of DRAM banks. Collected power traces were input to a compact 3D thermal model [109] which first converts the floorplan power to thermal grid power and calculates the temperature based on power density and thermal coupling of each grid. The calculated steady-state temperatures are then used to calculate refresh rates of different dies (ranks) and channels which is fed back to USIMM to calculate the updated performance results. For temperature calculations, we assumed silicon-based interposer with negligible thermal coupling between the core and DRAM dies, this is line with [127] which shows that this coupling is low as long as the distance between the two dies is greater than 10mm.

For microfluidics cooling simulations, we used parameters given in Table 12 which are directly taken from [109]. For air cooling, a high end server class heat sink is considered and the channel height between subsequent dies in the stack is reduced; see Table 12.

7.2 Thermal Characteristics of a 3D System

As discussed earlier, one of the fundamental challenges of a 3D stacked system is to cool a high power dissipating die which is hidden several layers beneath the heat sink. Figure 56 a) gives thermal maps for the 3D stack explained earlier with CPUs operating at a frequency of 2.4GHz. Two important observations can be made from the figure.

Table 12: Cooling parameters

Parameters	**Values**
Liquid Cooling:	
Pin fin height	300um
Pin fin diameter	100um
Width of thermal grid cell	200um
Height of thermal grid cell	200um
Effective heat transfer coefficient for the bottom of stack	2200W/m^2K
Ambient heat transfer coefficient for the top of stack (Hamb)	10W/m^2K
Ambient temp	20°C
Pumping Power	0.5W
Air Cooling:	
Effective heat transfer coefficient (Hamb)	100W/m^2K
Height between dies	100um

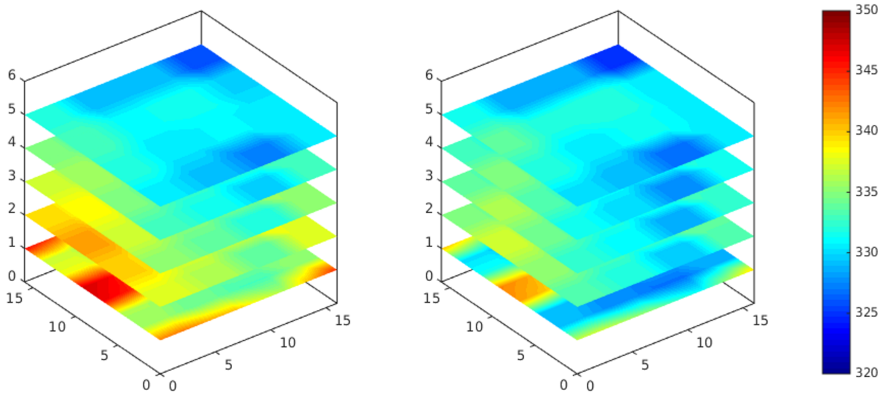


Figure 56: 3D Thermal map @2.4GHz with die separation of a) 300e-6m and b) 100e-6m

7.2.1 Higher Overall Temperature

The baseline many-core processor die dissipates a large amount of power increasing its temperature. In a 3D system, this die is at the bottom of the stack while the heat sink and the fans are connected to the top with multiple DRAM layers in between. This decreases the heat removal capability of the heat sinks increasing the temperature of the chips. Furthermore, since the processors, the caches, and the DRAMs are stacked, the overall power of the system is dissipated in a smaller area increasing power density and hence has a higher overall temperature as compared to conventional 2D or 2.5D designs. Furthermore, unlike 2D or 2.5D systems, the temperature of DRAM is induced by the activity of the underlying multicore die and the characteristics of the applications running on it.

7.2.2 High Variance within and across the Dies

The second observation to make is the high variation within and across the dies. Thus, the dies at the top will be cooler as compared to the dies at the bottom. The amount of variation across the dies depends upon the thermal coupling between the dies, which is a function of the distance between them, that is, the TSV height. A reduced height of 100um, as shown in figure 56 b), increases the thermal coupling decreasing variation and hence have higher temperature specially of the top dies even with the same heat sink. Variations within a die depends upon the application characteristics, processor floorplan, and memory access pattern across various DRAM channels.

7.2.3 Higher DRAM leakage

Higher temperature decreases the charge retaining capability of DRAM, increasing its leakage. Typically DRAMs restore their charge by periodically reading and writing the DRAM cells, a process known as DRAM refresh. Lower charge retaining capability means refresh operations at a faster rate increasing refresh power and performance penalty. We will analyze the refresh problem of 3D memories in detail in the next chapter.

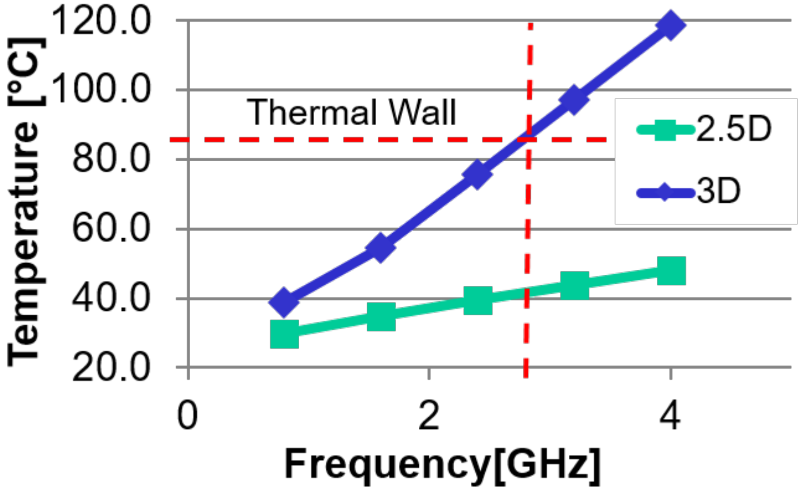


Figure 57: 3D thermal wall. Temperature of 3D stacks rise sharply with activity of cores.

7.2.4 3D Thermal Wall

Figure 57 gives the rate of increase of temperature in 3D and 2.5D stacks by increasing the frequency of the cores. It indicates a linear increase in temperature for both the stacks. The increase in the 3D case, however, is much higher as compared to the 2.5D case, primarily because it's temperature is governed by the temperature of the underlying multicore die experiencing much wider range, whereas the temperature of the 2.5D stack is mainly induced by DRAM activities which occur at a significantly slower rate. This sharp rise in temperature prohibits the full potential of 3D stacks even though they have much higher bandwidth and thus capability to handle higher loads. In other words, 3D stacks hit the thermal wall at higher frequencies prohibiting performance scaling. It should be noted that the rise in temperature is caused by the higher activity of the cores which can occur due to various reasons, such as, higher frequency, complex cores, compute-intensive applications, power-hungry accelerators, etc. Furthermore, the increase in temperature decreases the charge retention capability of DRAM increasing refresh rate, and also has a significantly negative impact on DRAM lifetime and reliability. As a result, it is advised to use 3D stacks only for memory-intensive applications running at a lower frequency.

7.3 3D Reliability and Thermal Dependency

In the previous section, we established that 3D DRAM operates at much higher temperature than their 2.5D counterpart. We will later show that this rise in temperature has low impact on performance specially for memory-intensive applications. However, increased temperature in 3D may affect system reliability significantly. To understand how DRAM reliability is dependent on its operating temperature, we adopted a DRAM lifetime reliability model from Micron [102], where mean-time-before-failures (MTBF) is proportional to temperature and voltage acceleration factors, AF_T and AF_V,

$$MTBF \propto AF_T \times AF_V = e^{\frac{E_a}{k}\left(\frac{1}{T_o}-\frac{1}{T_s}\right)} \times e^{\beta(V_s-V_o)} \tag{3}$$

where E_a is the activation energy, and k is the Boltzmann's constant. T_o and T_s denote operation and stress temperatures, respectively. β is a process-dependent constant, and V_o and V_s are operation and stress voltages.

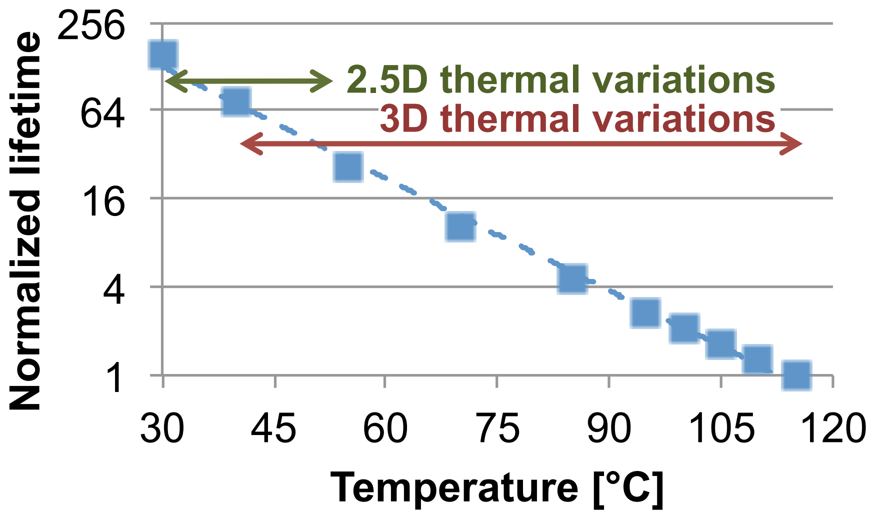


Figure 58: Impact of temperatures on DRAM lifetime reliability, and the extent of thermal variations in 2.5D and 3D processors. Lifetime reduces exponentially with temperature.

Figure 58 shows the correlation between operating temperature and the resulting DRAM lifetime. It can be seen that DRAM lifetime degrades very quickly with increased operating temperature.

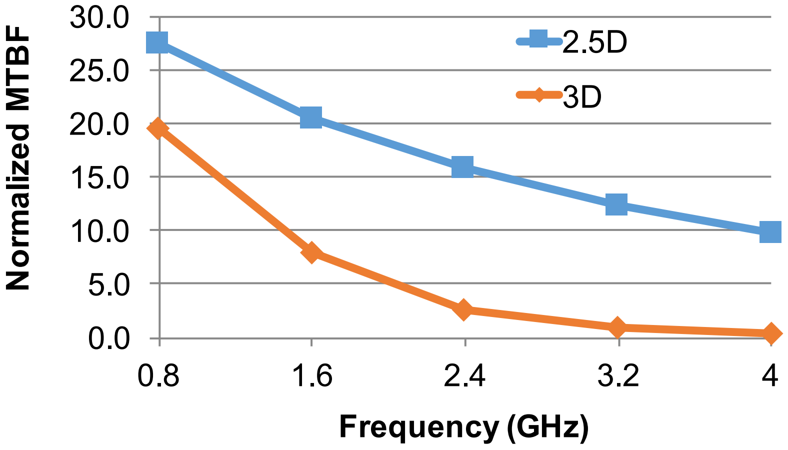
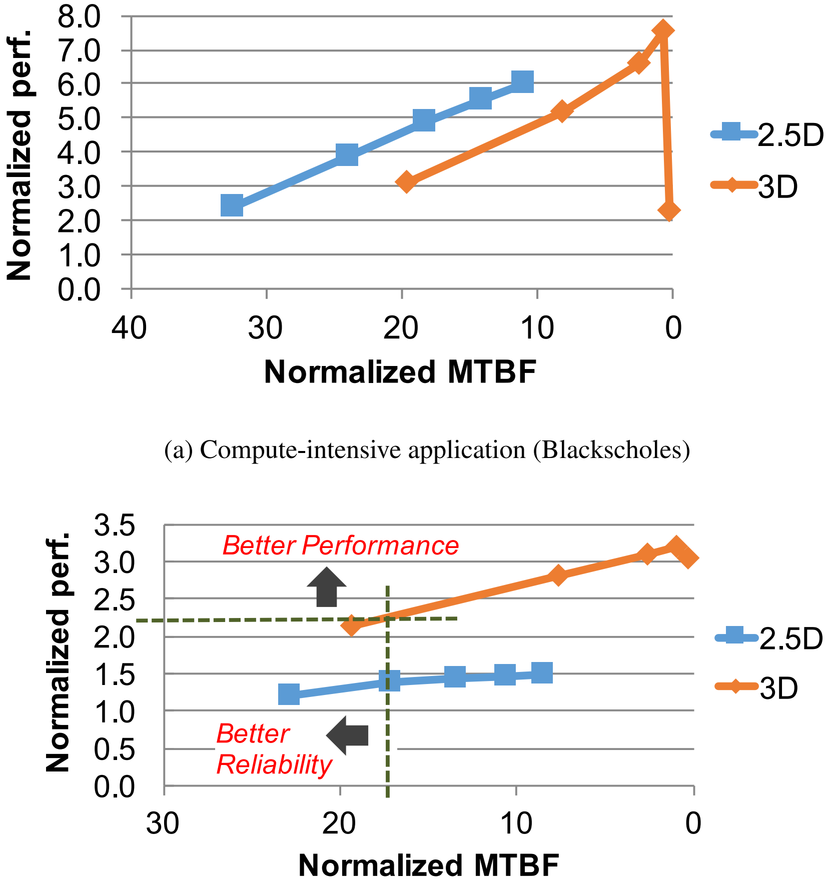


Figure 59: 2.5D and 3D lifetime at different operating frequencies. 3D lifetime degrades rapidly with frequency.

Using the thermal-reliability relation discussed above, Figure 59 shows the impact of increasing core frequency on DRAM lifetime reliability. The result is an average of multiple PARSEC/SPLASH benchmarks. Since the estimated operating temperatures of 3D stacks are greater than 2.5D stacks, 3D DRAMs exhibits much lower lifetime. Furthermore, 3D lifetime degrades quickly with increased frequency, which suggests that 3D systems should be operated at much lower frequency than their 2.5D counterparts.

7.3.1 Reliability-Performance Tradeoff

Figure 60a and 60b plot the performance and reliability of 2.5D and 3D-stacked processors for representative compute- and memory-intensive applications. For compute-intensive application, 2.5D designs are better. They not only have better reliability characteristics but also have higher performance than 3D at the same reliability design point. Performance can further be improved by operating at higher frequencies. For memory-intensive applications, on the other hand, 3D stacked designs are better based on their superior performance. Reducing frequency to improve lifetime reliability still achieves better performance relative to comparable 2.5D design even at higher frequencies.



(a) Compute-intensive application (Blackscholes)

(b) Memory-intensive application (Canneal)

Figure 60: Performance and reliability tradeoff between 2.5D and 3D-stacked DRAM processors. Compute-intensive applications has better tradeoff with 2.5D while memory-intensive applications prefers 3D memory systems.

7.3.2 Throughput-Lifetime Product

Song et al. [91] suggested using a metric of *throughput-lifetime product* (TLP) to quantify reliability-performance tradeoffs. Figure 61 and 62 illustrate the TLP of various applications at different frequencies for 2.5D and 3D stacked memory systems, respectively. In general, TLP of 3D design is higher at low frequency, specially in the case of memory-intensive applications. However it decreases rapidly with increasing frequency due to lower lifetime reliability. On the other hand, the variation in TLP in a 2.5D stacked system is much lower, due to a much smaller variation in temperature with varying core frequencies. In fact, for most applications (except canneal), small increase in frequency increases the

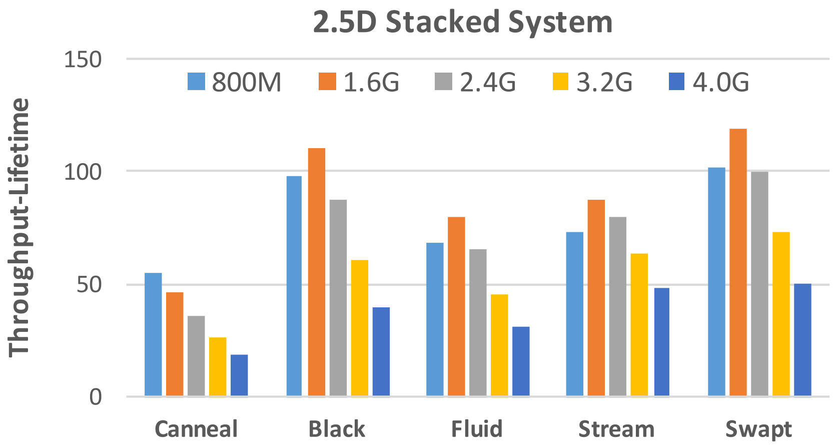


Figure 61: Throughput-lifetime product of various applications with various frequencies. 2.5D stacked systems exhibit much lower variation in TLP with changing frequency.

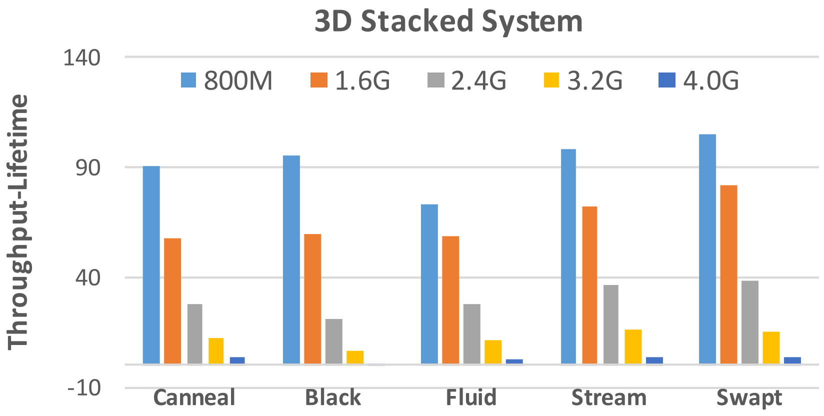


Figure 62: Throughput-lifetime product of various applications with various frequencies. 3D stacks show higher variation in TLP.

overall TLP, showing 2.5D stacks can be operated at higher frequencies. To conclude, the 3D design is suited for cases of memory-intensive applications operating at low frequency. For all other cases, the 2.5D system achieves better performance-reliability tradeoff.

7.4 Impact of Microfluidics Cooling

Liquid cooling has been proposed as a viable solution to keep the temperature of 3D stacks low. Various type of liquid cooling solutions have been proposed. Bakir et. al proposed pin-fin enhanced microgap cooling [123] that allows the fluid to flow next to the chip increasing its effective heat transfer coefficient and cooling capability. Lower temperature increases the operating range of the overall system allowing sustained performance scaling. This section attempts to quantify the advantages of using microfluidics technology for continued performance scaling.

7.4.1 Thermal Maps with and w/o Microfluidics Cooling

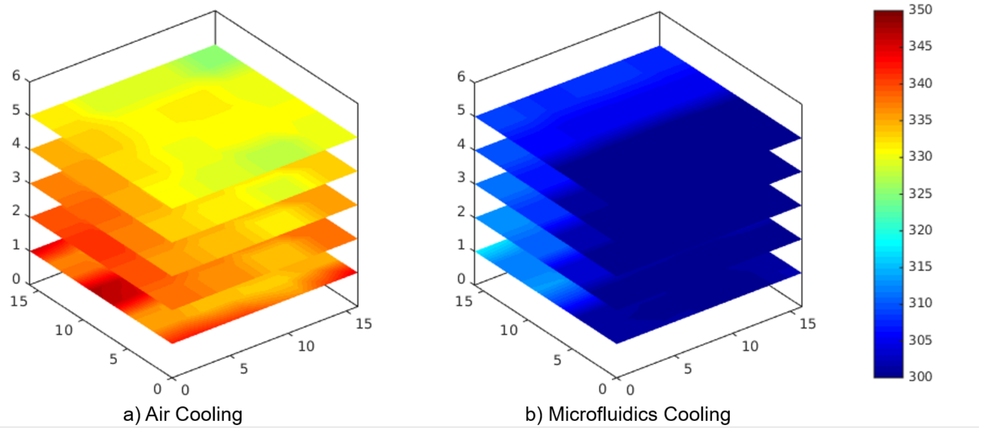


Figure 63: Heat-map of blackscholes @2.4GHz with and w/o microfluidics. Microfluidics cooling keeps the temperature very nominal.

Figure 63 gives the heat map of blackscholes at a particular time instant with and without microfluidics cooling in a 3D stacked DRAM. The air cooled system has a high variation in temperature. The bottom core die dissipates large amount of power and its temperature reaches almost 350 Kelvin. The top die is closer to heat sink and thus has higher cooling capacity and operates at a much lower temperature; around 320 Kelvin in this particular case. The microfluidics cooled system, on the other hand, is much cooler with maximum temperature below 320 Kelvin. We assume that microfluidics is only applied at the logic core die. The variation in temperature across the dies is also very minimal. Since the coolant flow from one direction to the other, the temperature at one side is slightly lower than the temperature at the other side. However, the variation is low. With such low operating temperatures using microfluidics cooling technology, the refresh rate can be kept at a minimal leading to significant reduction in refresh power and performance penalty. Furthermore, programs can be run at much higher frequency without heating the system significantly.

7.4.2 Performance Scaling with and w/o Microfluidics

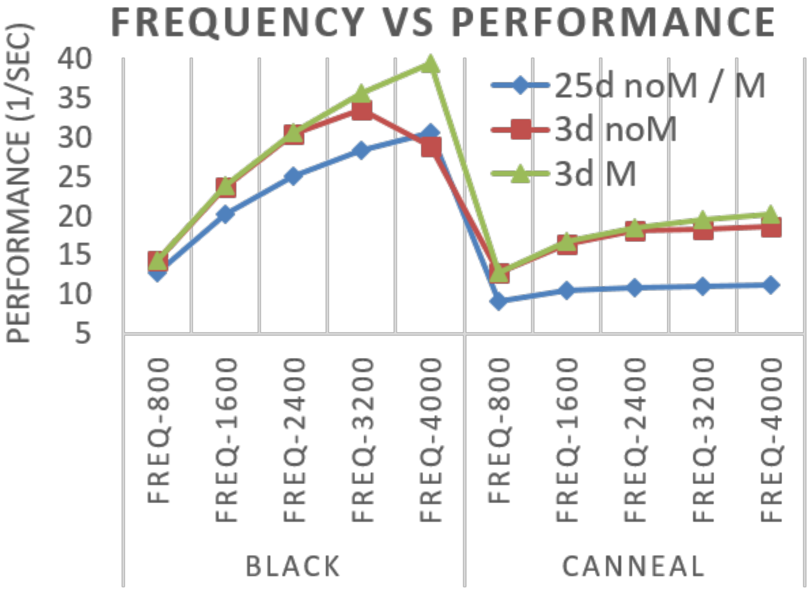


Figure 64: Frequency vs Performance. Microfluidics cooling allow continued performance scaling specially for compute intensive applications.

We analyzed the impact of microfluidics technology on performance scaling using various scaling parameters. These techniques are known to increase power of the cores and hence temperature. We assumed that we have infinite power budget, and memory bandwidth and temperature are the main performance limitations. Microfluidics cooling will keep the temperature low, hence have lower refresh rate and thus more bandwidth availability at much higher operating power. We analyzed performance scaling with three different knobs. 1) by increasing frequency, 2) by increasing the number of active cores, and 3) by increasing the core complexity (super-scalar width and ROB size). At low frequencies (Figure 64), the performance with microfluidics remain similar to the case with no microfluidics technology. Here, $3d - noM$ indicates air cooling and $3d - M$ represents 3D stacks with microfluidic cooling. At higher frequency, the impact of higher refresh rate in the no microfluidic solution starts to dominate. However, microfluidcs keeps the temperature low and hence allows for continued performance scaling with frequency.

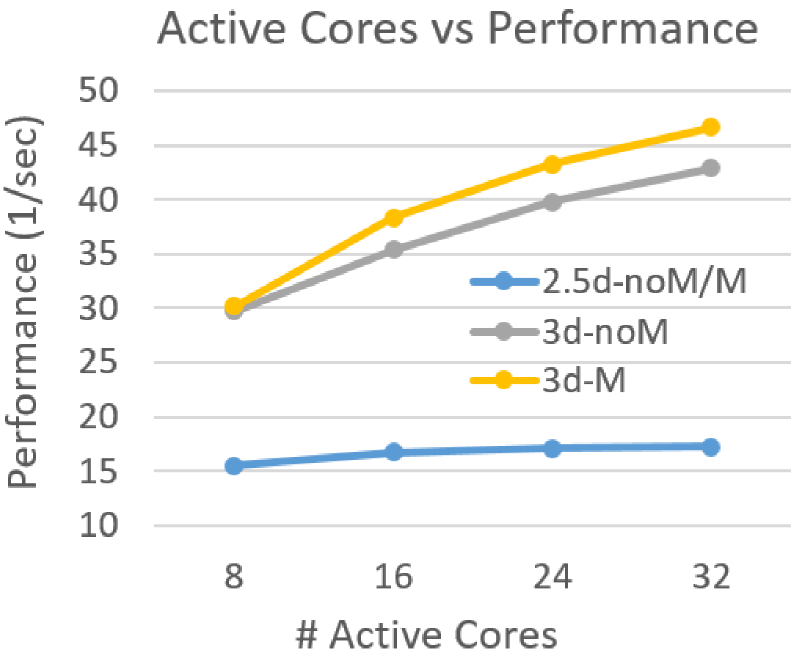


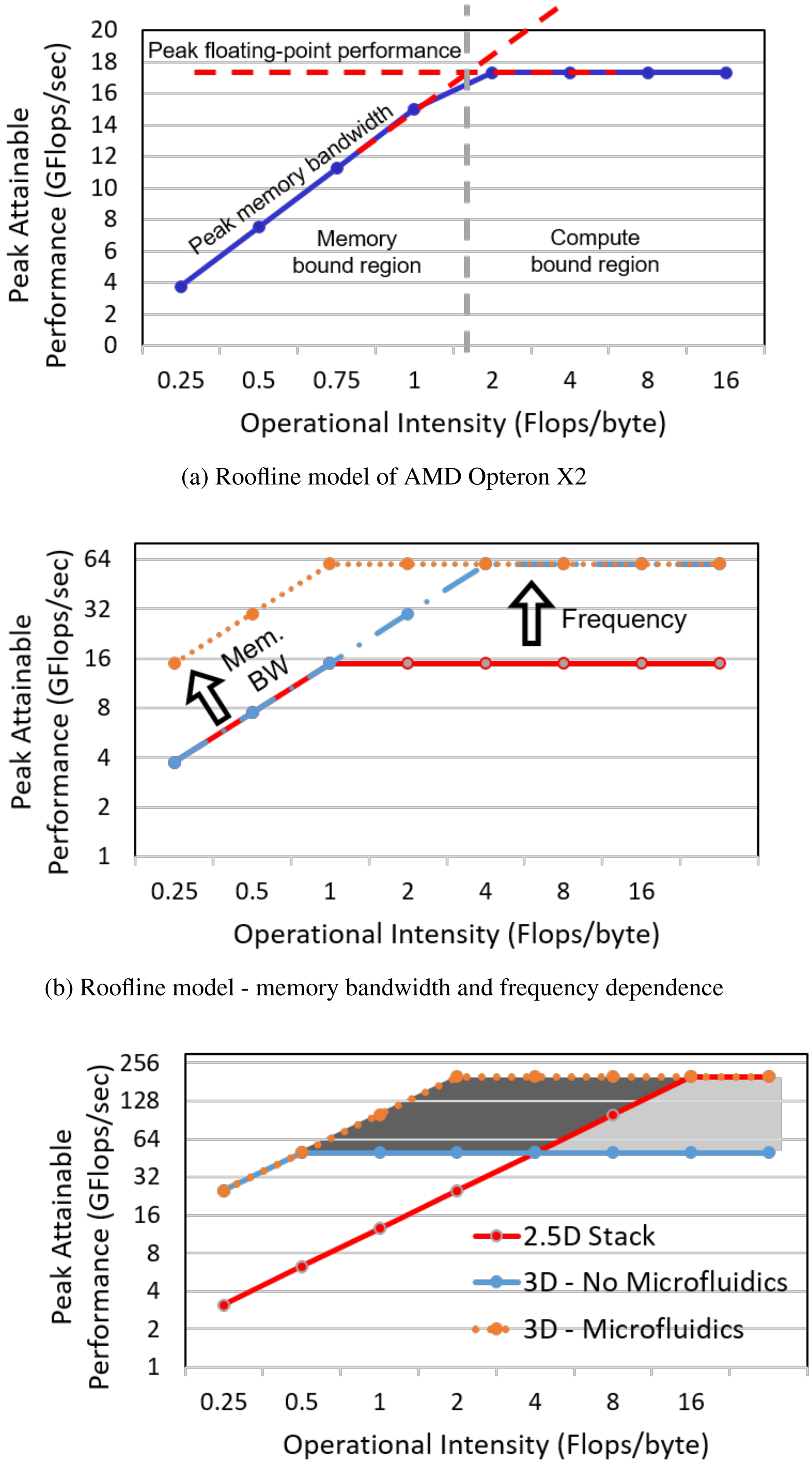
Figure 65: Active Cores vs Performance. Microfluidics cooling supports core scaling in the logic die.

The impact of microfluidics is even higher with increasing core count. Larger number of cores increases the pressure on DRAM bandwidth and hence makes it more sensitive to refresh rate. Even more, it increases the power of the logic die significantly. Figure 65 plots the performance of canneal with and without the microfluidics technology. The improvement using microfluidics is 6-7%. For compute intensive applications, performance improvement is even higher. Similar results can be obtained by increasing the complexity of cores (not shown), which increases the ROB size, and hence puts a significant load on DRAM bandwidth making it sensitive to temperature.

7.4.3 Impact of Microfluidics using the Roofline Model

A common way to understand the performance of a computer system is to look at the roofline model [114]. The roofline model relates a system's overall performance with operational intensity of applications and memory bandwidth in a 2D graph. Figure 66a shows the roofline model for a 2.2GHz Opteron X2 (taken from the original roofline paper [114]). Operational intensity of an application, which is the x-axis of the graph, is defined as the number of operations or flops per byte of DRAM traffic. The y-axis plots the maximum attainable performance. The model has two distinct lines; the compute line, valid at high operational intensity, plots the peak floating point performance of the system. The performance of an application cannot be higher than this (compute-bound). And the memory line, valid at low operational intensity, determines the peak memory performance for memory intensive applications. The attainable performance can be obtained as the minimum of the two as shown in Figure 66a. The peak performance line is a function of the frequency of the cores. It moves up and down based on the operating frequency (Figure 66b). Similarly, the slope of the memory line is a function of peak memory bandwidth of the system. On a log scale, this is equivalent to shift in the memory line as shown in Figure 66b where the intercept determines the peak memory bandwidth.

Figure 66c shows the roofline model of a 3D stack with peak memory bandwidth of 160GB/s compared with a 2.5D stack with memory bandwidth of 40GB/s. Due to higher bandwidth, a 3D memory system can achieve higher performance at a much lower operational intensity indicating its usefulness for memory-intensive applications. However, thermal constraints limits operation at higher frequencies. Thus, the peak performance is well below the maximum performance that can be achieved by a 2.5D stack running at



(a) Roofline model of AMD Opteron X2

(b) Roofline model - memory bandwidth and frequency dependence

(c) Roofline model of 2.5D and 3D stack with and without microfluidics

Figure 66: The roofline model with thermal constraints and microfluidics cooling. Microfluidics cooling allow both compute and memory intensive applications to run at higher frequencies.

higher frequency, specially for applications with high operational intensity (aka compute-intensive applications). It can be deduced that although 3D stacks have higher bandwidth, the utilization of that bandwidth is limited only to memory-intensive applications running at low core frequencies.

As shown in Figure 66c, microfluidics cooling removes the thermal constraints allowing 3D stacks to operate at same peak core performance as a 2.5D stack. In the figure, the dark shaded region constitutes of the applications that were not feasible for both 2.5D and 3D stacks due to their moderate operating intensity and higher frequency. With microfluidically cooled 3D stacks, these applications are not only feasible but can achieve maximum performance. Example of such applications include memory-intensive accelerators like neural networks and GPUs running at a very high frequency. Figure 67 shows the utilized bandwidth vs temperature for canneal running at different frequencies with and without microfluidics technology. If we assume a thermal cap of 70C, we can see that the maximum utilized bandwidth that can be achieved at this temperature is almost 50% higher in the case of microfluidics cooling instead of an air-cooled system.

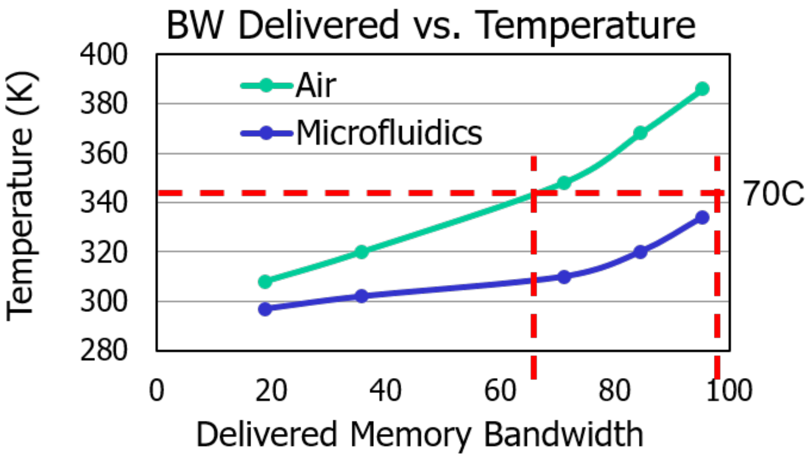


Figure 67: Utilized 3D bandwidth with and without microfluidics cooling

The light shaded region of Figure 66c constitutes compute-intensive applications and core configurations such as complex out-of-order cores that were only feasible with 2.5D stacks. We conclude by pointing out the fact that although 3D stacking has removed the memory bandwidth bottleneck for a number of applications, thermal constraints has put a limit on the achieved utilization of this bandwidth both for memory-intensive applications running at higher frequencies and compute-intensive applications. Microfluidics cooling allow us to fully utilize the bandwidth available in 3D memories and pushes its envelope to newer domains.

7.4.4 Challenges and Future Potential of Microfluidics Cooling

Sarvey et al. [83] explored different types of microfluidics cooling solutions. As discussed earlier, we can provide pin-fin enhanced heat sink to all the dies or only to the high power generating core die. The first solution has higher cost and power but can work with high power dissipating solutions, such as 3D FPGAs. The second solution works in the cases where there is a clear distinction between high power and low power generating dies. Another solution can be to pump coolant only through the interposer using similar micro pinfins. Since, the logic die is at the bottom, large amount of heat can be reduced by cooling the interposer layer only. However, the design has two challenges. First, it will increase the thickness of the interposer which will result in longer TSVs with increased losses. Second, thermal coupling between the processor and memory die will still increase the temperature of the memory dies hurting performance. The paper proposes thermal resistant air gap between high power dissipating core dies and low power dissipating memory dies to reduce the thermal coupling between them. Another challenge that is discussed briefly in the paper is the impact of thermal coupling between different stacks placed on an interposer, specially if the horizontal distance among them is small. As discussed previously, yet another challenge with pin fin based cooling is the distribution of the pins. Since TSVs can only be placed inside the pins, thermal and electrical design considerations have to be taken simultaneously as proposed in [117].

We conclude by pointing out that our analysis is the first attempt to understand the impact of high-end cooling solutions, such as microfluidics, on the performance of a 3D stacked system. However, the analysis has a number of limitations. We have not explored the impact of different cooling design parameters and various implementation options in this study. Furthermore, the study uses same application to run all the cores. This means our workload was either memory intensive or compute intensive. On the other hand, mixing various compute- and memory-intensive applications simultaneously will give an interesting behavior where the overall temperature is higher due to compute-intensive applications while the memory bandwidth requirement is also high due to the memory-intensive applications. This case incurs penalty of both scenarios, higher temperature with higher memory bandwidth demand. We plan to explore these options in future.

7.5 Concluding Remarks

In this chapter, we analyzed the thermal problems of 3D memory. We identified the rise in temperature and its variance within and across the dies as the two major thermal bottlenecks leading to increased DRAM leakage and hence higher refresh rate. We also quantified the impact of high temperature on DRAM's lifetime reliability. We showed that newer cooling solutions, such as microfluidics cooling, can help alleviate 3D stacks temperature problem and allow it to run new compute-intensive applications and operate at much higher frequencies utilizing the full potential of the high bandwidth of 3D memories. In the next chapter, we exploit the variance in stack DRAM's temperature to propose variable-rate refresh that not only reduces 3D stacks refresh power but also increases its operating range by allowing DRAMs to function correctly at significantly higher temperatures.

CHAPTER 8

VPREM - VARIABLE-RATE PER-BANK REFRESH MANAGEMENT FOR 3D STACKED SYSTEMS

3D stacks provide greater memory bandwidth but DRAM dies suffer from higher operating temperatures induced by underlying multicore die and application characteristics. The major problem is a lower heat removal capability which increases their temperature and requires larger number of rows to be refreshed at significantly higher rates. Increased refreshing time decreases memory bandwidth availability and increases refresh power prohibiting continued performance scaling. Another problem with 3D stacks is the high variance in temperature within and across the dies. Variance across the dies is created by the variation in heat removal capability of conventional air-cooled heat sinks. Variance within the dies is caused by the uneven activity of the underlying multicore die heating DRAMs differently at different banks and channels. This chapter exploits the variation in 3D DRAM temperature and propose variable-rate per-bank per-channel refresh allowing different channels and banks to be refreshed at different rates based on their operating temperatures. It allows the DRAM to operate at higher temperatures tolerating hot-spots in the memory while keeping the refresh power in check.

8.1 DRAM Refresh Problem

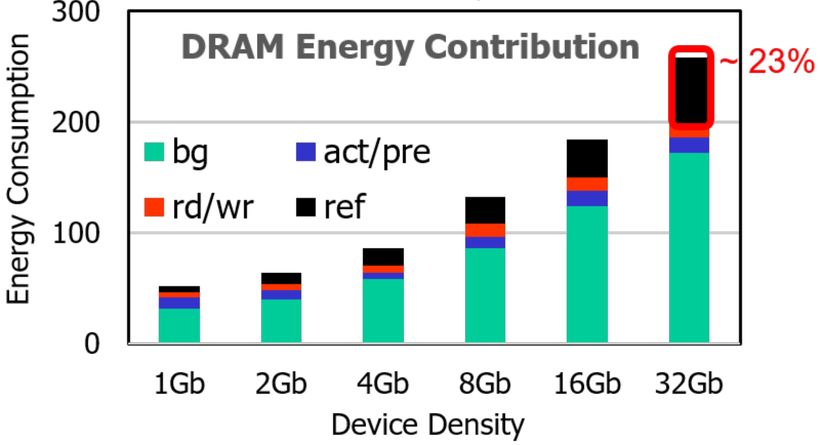
DRAM cells are leaky, that is, they lose their charge after a certain amount of time. To restore the charge all the DRAM cells are read and written again periodically; a process known as DRAM refresh. The frequency of DRAM refresh depends on the retention time characteristics of the cells which is a function of temperature. Typically, for temperatures below 85°C, DRAMs are refreshed at a period of 64ms and at double the rate for every 10°C rise [60]. Since a DRAM has a large number of rows, the refresh operation is staggered, typically in 8K intervals [8]. Each refresh operation refreshes multiple number of rows (equivalent to the total number of rows divided by 8K). The time taken to perform a refresh operation *t_RFC* depends on the number of rows per refresh which is increasing rapidly with higher density DRAMs. In conventional DDRx, a refresh operation is performed on all the banks simultaneously making them unavailable during the refresh operations. This leads to unavailability of DRAM bandwidth during refresh. Furthermore, with lower device feature size across technology generations, DRAM reliability becomes a major challenge leading to the proposal of refresh rate of 32ms in modern devices [96]. Table 13 give *t_RFC* values for different DRAM sizes along with the percentage of time DRAM is unaccessible [51]. With more denser DRAMs, refresh can lead to DRAM bandwidth unavailability for almost one third of the time for 64Gb DRAMs.

Table 13: Refresh rate with various device sizes

DRAM Density	**t_RFC**	**BW Loss @64ms**	**BW Loss @32ms**
512Mb	90ns	1.15%	2.3%
1Gb	110ns	1.41%	2.82%
2Gb	160ns	2.05%	4.1%
4Gb	260ns	3.33%	6.66%
8Gb	350ns	4.49%	8.98%
16Gb	480ns	6.15%	12.3%
32Gb	640ns	8.21%	16.42%

The second major problem with refresh is its high power consumption, specially with large capacity DRAMs. As a general rule of thumb, refresh power is linearly proportional to DRAM capacity. Furthermore, the DRAM capacity requirement of large data centers keeps on increasing but its utilization of individual DRAM components reduces. With lower DRAM utilization, the power to do the actual read/write operations decreases, making background and refresh power the major components of DRAM power. Figure 68 gives a breakdown of DRAM energy consumption with various device sizes (Taken from [12]). Activate / precharge and read / write power is related to the actual DRAM activity and depends on the number of accesses and their access patterns. Background and refresh power, on the other hand, can be considered as the static or idle power of the DRAM.

It can be seen that both background and refresh power increases rapidly with increasing device size, with refresh taking almost 23% of the total DRAM power in 32Gb devices. Background power can be reduced by utilizing various low power modes during the low activity modes of DRAM. However, reducing the refresh power remains a challenge and can be reduced mainly by reducing the number of actual refresh operations, the goal of this chapter.



Taken From: I. Bhati, “Scalable and energy-efficient DRAM refresh techniques”, PhD Thesis, 2014.

Figure 68: DRAM power breakdown with various device sizes [12]

8.1.1 DRAM Refresh Dependency on Temperature

The retention time characteristics of various DRAM cells varies widely. The rate of 64ms is a conservative value designed for the weak cells to operate at high temperatures. Some recent works have pointed towards decreasing the refresh rate for lower temperatures [82]. A feature called thermal-compensated self-refresh [103] (TCSR) is already provided in modern DDRs which reduces the refresh rate based on the device temperature in self-refresh mode. However, this mechanism is based on internal DRAM refresh counter in self-refresh mode and temperature cannot be regulated from the outside. We propose thermal-regulated variable rate refresh across different DRAM banks and channels, controlled by the memory controller. We extrapolated temperature vs. refresh rate curve using different current values in TCSR mode as shown in Figure 69.

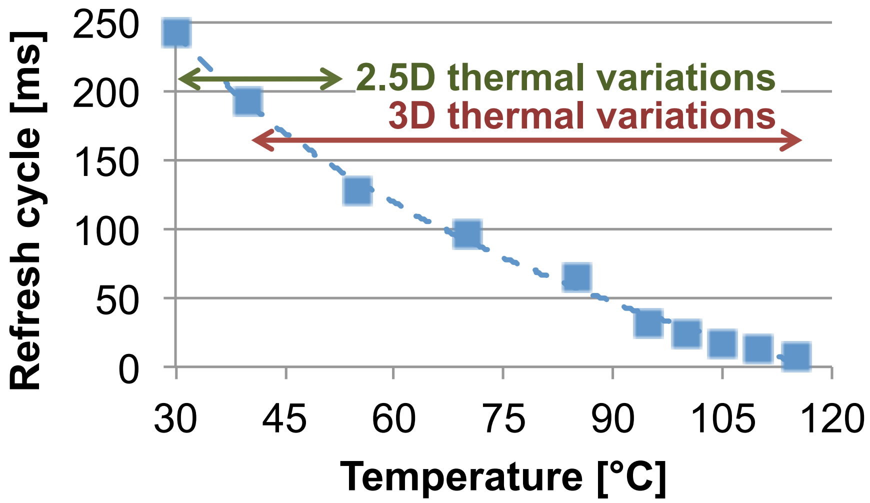


Figure 69: Refresh cycle time for DRAM under different operating temperatures [82]. The graph is made by looking at current values of temperature-compensated self-refresh

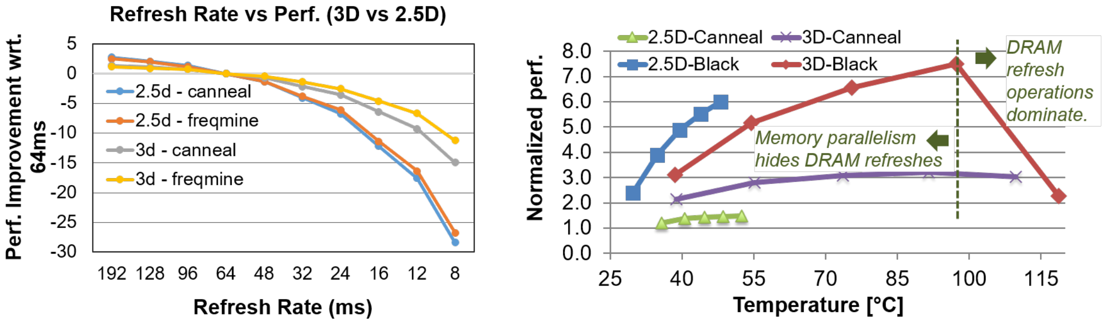
8.2 Impact of 3D Characteristics on Refresh

This subsection explains the impact of 3D stacking technology on performance and power overheads of DRAM refresh. We will focus on the differences related to DRAM refresh in 3D compared to the conventional 2D or 2.5D systems.

8.2.1 Reduced Refresh Performance Penalty

The first observation to make is that high parallelism in 3D memory systems reduces the performance penalty of DRAM refresh operations. Figure 70a gives the refresh rate vs normalized performance for a memory-intensive and a cpu-intensive application for a 3D-stacked memory with 16 channels and a 2.5D-stacked memory system with four memory channels as explained in section 3.2 and 7.1, respectively. The performance is minimized to the baseline 64ms case. The simulation parameters are the same as explained in section 7.1. It can be seen that higher refresh rates have smaller impact on 16-channel 3D systems as compared to 4-channel 2.5D systems mainly because of the following two reasons. First, it is because of the smaller DRAM bank size leading to lower refreshing time and thus lesser performance penalty. Second, it is because of the higher channel-level parallelism available in 3D memories. Higher parallelism decreases the probability that a DRAM access will collide with the refreshing channel leading to unavailability of memory bandwidth and reduced performance. Furthermore, with the advent of per-bank refreshing [13], bank-level parallelism can also be exploited to further reduce the performance penalty of DRAM refresh.

8.2.2 Thermal Limit Affecting Performance



(a) Performance vs refresh rate

(b) Performance vs temperature

Figure 70: Correlation between performance, power, and temperature of 3D vs. 2.5D stacks

We increase the frequency of cores from 800MHz to 4GHz. We observe a rapid linear increase in temperature with increasing frequency for both 2.5D- and 3D-stacked memory. The increase in the 3D case is much higher primarily because it is governed by the temperature of the underlying multicore die. Figure 70b shows the impact of DRAM temperatures (and corresponding refresh rates) on performance. The DRAM bandwidth of memory-intensive applications is already saturated. Higher temperature by increasing frequency does not improve performance significantly. Any improvement in performance is mitigated by the increased refresh rate reducing memory bandwidth availability. The performance of compute-intensive applications, on the other hand, increases with frequency. Even though the 3D design requires more frequent DRAM refresh operations than the 2.5D design, channel-level parallelism in the 3D design effectively hides DRAM refresh operations. Thus, higher operating temperature and refresh rate has relatively minor impact on the overall performance. However, at very high temperature and frequencies, refresh operations dominates the DRAM activity allowing the performance to degrade abruptly. Thus, for compute-intensive applications, performance scaling by increasing the frequency of cores is limited by refresh operations or, in other words, by the temperature of the DRAM.

8.2.3 Correlation Between Performance and Refresh Power

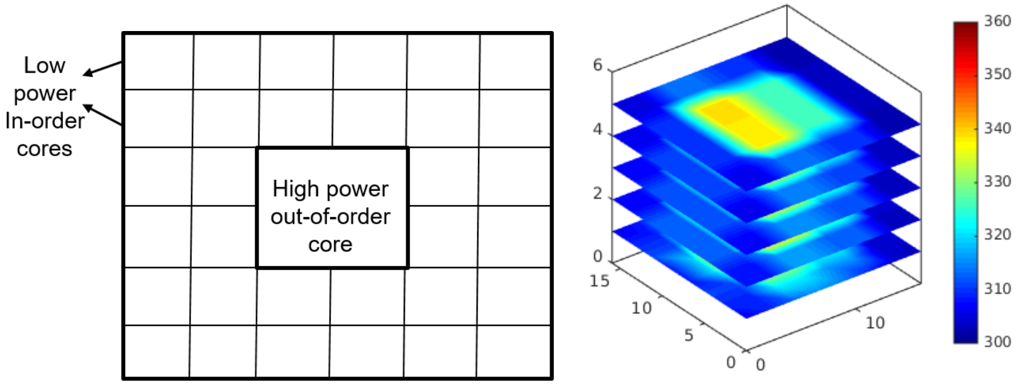
As mentioned earlier, 3D stacked DRAMs operate at higher temperature requiring DRAM refresh to be performed at a higher rate, thus increasing refresh power. Furthermore, in conventional DRAM systems, refresh power is directly proportional to the capacity of the DRAM. System operating conditions like the activity of the cores or activity of the DRAM itself has negligible effect on refresh power. In fact, during high DRAM activity, e.g, with high-bandwidth applications, refresh power constitutes a very small component of the overall DRAM power as the activate/precharge and read/write energy starts dominating. This trend, however, changes with 3D stacked DRAMs. In 3D DRAMs, the temperature of a DRAM channel and thus its refresh rate and power directly depends upon the activity of the cores placed right below the channels. Thus the compute-intensive applications, which dissipates significantly more heat in the core die because of its higher utilization, has much larger stack temperature than the corresponding memory-intensive applications, even if they have lower activity in the DRAM itself. Higher temperature means higher refresh rate and thus higher refresh power. Hence increasing the performance by increasing the frequency or complexity of the cores is directly proportional to increasing refresh power which may become a limiting factor of performance scaling. Exploiting the variation in DRAMs temperature to reduce the refresh power will directly impact performance scaling.

8.2.4 Summarizing the Idea

The summary of the discussion in the previous two sections is given in Table 14. In short, DRAM refresh has smaller performance penalty in 3D stacked systems as compared to their 2D or 2.5D counterparts. However, refresh power in the active mode becomes a bigger challenge because of 3D DRAM's higher operating temperature, its dependence on the activity of the cores, and its correlation with system performance. In this work,

Table 14: Refresh related differences in conventional and 3D memories

	Conventional Memory	3D Memory
Performance penalty of refresh	**High**	**Low**
Thermally-limited DRAM performance	**No**	**Yes**
Refresh power % vs system performance	**Inverse correlation**	**Direct correlation**
Refresh power sensitivity to core's activity	**Low**	**High**
Temperature and refresh rate variation	**Low**	**High**
Exploitation of variation in refresh rate	**Low potential**	**High potential**



(a) Core Die Floorplan

(b) Heat Map

Figure 71: An example floorplan and thermal profile of a heterogeneous 3D stacked system

we propose to exploit the variation in DRAM temperature to reduce the overall refresh power and allow some parts of the DRAM to operate at much higher temperature than the regular operating range. Next, we present a motivating example showing how variation in temperature can be used to reduce refresh power.

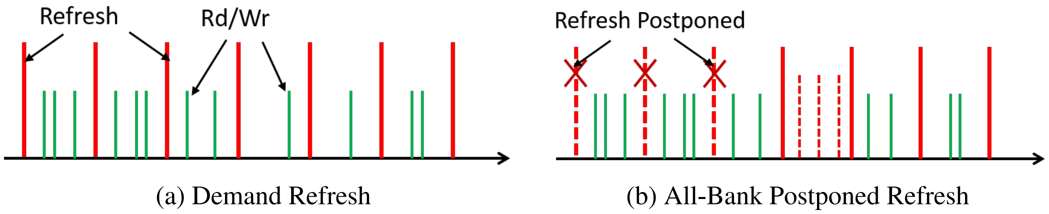
8.3 A Motivating Example

Consider an example of a heterogeneous system with a single out-of-order and multiple in-order cores in the baseline logic die of a 3D stacked memory system as shown in figure 71a. The die area is divided into a 6x6 tiles with the high-power out-of-order core taking 4x area of a low-power in-order core. The in-order cores will be waiting mostly for the read/write requests to get serviced from memory. Thus, their activity and power will be low. On the other hand, the activity, thus the power and the heat generated by the out-of-order core will be significantly higher as compared to the power generated by the in-order cores. Figure 71b gives the thermal profile of the system. The high power of the out-of-order core will increase the temperature at the center of the baseline logic die, which will be propagated both horizontally and vertically because of the thermal coupling within and across the dies. However, this coupling will be strong near the hotspots and low away from it. Thus, we will get a highly variable temperature map of the overall system. The difference in temperature reaches almost 70C between the hotspot logic and the corner of the top die. The temperature gradient between various parts of the DRAM reaches almost 50C. Thus, the difference in charge retaining capability of DRAM cells in hot and cold regions will be very high. Furthermore, it is very likely that the memory data for in-order cores is mapped closer to the core in the sub-stacks above it in order to reduce the traversals and latency in the horizontal direction. Note that the commonly used first touch OS allocation policy will try to keep the data local to the requesting cores. Thus, high percentage of memory requests of the in-order cores may come from the cold regions stacked directly on top. Refreshing the cold regions at the same rate as hot regions not only increases refresh power, it will unnecessarily penalize the latency of requests coming from the in-order cores, thus reducing their performance. Furthermore, by refreshing only the hot regions at a much faster rate and saving refresh power for the cold regions, we can increase the operating thermal range of the overall system. Hence, we propose thermally-compensated variable-rate per-channel per-bank refresh that reduces DRAM refresh power and increases its operating thermal range.

The discussion aboves supports the following key point. Refreshing all DRAM channels and dies based on the hottest operating temperature unnecessarily increases total execution time and energy degrading both performance and power. For example, in the previous example, if we refresh the whole DRAM based on the operating temperature of the die right above the hotspot, we will either refresh at a very fast rate incurring significant refresh performance and power penalty or we will reduce the frequency of the out-of-order core reducing temperature and performance both. In such a scenario, allowing the channels and banks close to the hotspot to refresh at a much higher rate while keeping nominal refresh for others allows us to keep refresh power low as well as keep the operating frequency of the out-of-order core high, thus maintaining high performance.

8.4 Various Refresh Management Aspects

Next, we will discuss various aspects of modern refreshing techniques that is necessary to understand the advantages and disadvantages of variable-rate per-bank refresh.



(a) Demand Refresh (b) All-Bank Postponed Refresh

Figure 72: Advantages / disadvantages of postponed refresh

8.4.1 Demand vs. Opportunistic vs. Postponed Refresh

As explained earlier, a common way to perform refresh is to divide a refresh cycle into 8K intervals with one refresh command sent per interval, referred here as *demand refresh*, and shown in Figure 72a.

The interval length, denoted by DRAM timing parameter *t_REFI* determines the refresh rate, e.g., 7.8us for a refresh period of 64ms. However, forcing refresh after every 7.8us can delay critical reads if it collides with the refresh operation. A common approach to solve this problem is to perform refresh opportunistically, that is, send a refresh command whenever the DRAM rank or channel is not busy serving any reads or writes while maintaining a refresh cycle of 64 ms. This mechanism has the potential of performing many refresh operations in a short interval of time. Similarly, this mechanism allows postponing the refresh operations during high bandwidth periods. However, it can also increase the time between two subsequent refreshes going to the same memory location. Although some slack can be tolerated due to higher retention time in most cases, JEDEC allows refreshes to be postponed only slightly. For example in DDRx devices, eight refresh commands can be postponed or sent in advance to allow for more critical read/write requests to be serviced earlier without any delay. However, if eight refresh commands are not sent in $8 * t\_REFI$ window, the memory controller stops servicing requests and forces all the postponed refreshes to be completed in a burst as shown in Figure 72b. We called the scheme where refreshes are performed opportunistically with fall back to a burst period in case of incompletion as *opportunistic refresh*, while the scheme in which refreshes are done only in a burst after every *8*t_REFI* without taking advantage of opportunistic refresh is referred to as *postponed refresh*. We will later show the challenges and advantages of opportunistic and postponed refreshes with our variable-rate per-bank refresh management (VPReM) scheme and provides a few solutions to overcome the challenges.

8.4.2 Fine- vs. Coarse-Grained Refresh

As mentioned earlier, a DRAM refresh cycle generally consists of 8K intervals. However, the number of rows in a DRAM are much higher than 8K, (e.g., 64K rows in each bank with a DRAM size of 4GB and a row buffer size of 8K). A refresh operation divides all the rows into 8K groups and refreshes one group or multiple rows simultaneously at each interval (four per bank in the previous example). The time for each refresh operation t_RFC is dependent on the number of rows being refreshed simultaneously. With large capacity DRAMs, this number is on the rise, hence increasing the value of t_RFC for modern DDRs. Read and write requests have to wait longer for their turn while a channel is being refreshed. To overcome long waiting time because of high t_RFC values, DDR4 [94] introduces the concept of fine-grained refreshes. The idea is to refresh a smaller number of rows per refresh command but with more frequent refresh operations, such as in 16K or 32K intervals for 2x and 4x modes, respectively. 1x refresh is a direct extension of DDR2 and DDR3 refresh: each refresh command takes t_RFC ns, and it must be issued every $t\_REFI = 7.8us$. 2x and 4x modes require that refresh commands be issued twice and four times as frequently at a rate of 3.9us and 1.95us, respectively.

In the next sections, we will discuss the challenges and advantages of two other refresh management aspects (i.e., per-bank vs. all-bank refresh and fixed-rate vs. variable-rate refresh) in more detail.

8.5 Per-Bank Refresh

In commodity DRAMs, a refresh is performed at a rank-level, that is, a refresh operation is performed on all the banks simultaneously making the whole rank unavailable during the refresh operations, generally referred to as an all-bank refresh. This decreases the total memory bandwidth availability of DRAM, which is declining with the rising value of t_RFC. More recent LPDDRs, however, have enabled a new mode called per-bank refresh, performing refresh one bank at a time, allowing other banks to be accessed simultaneously while one of the bank is unavailable performing the refresh operation. This leads to refresh being performed in parallel with read/write accesses reducing the bandwidth loss problem during refresh. Let's call the command sent to perform all-bank refresh as REF_{ab} and the command sent to perform per-bank refresh as REF_{pb} The time taken to perform refresh of a single bank t_RFC_{pb} is much lesser than the time taken to perform all-bank refresh t_RFC_{ab}. However, the frequency of REF_{pb} is NUM_BANKS times higher than the frequency of REF_{ab}. Thus the command bandwidth of per-bank refresh is much higher than its all-bank counterpart. We next present a few advantages and challenges with per-bank refresh and explain how we overcame these challenges in our design.

8.5.1 Advantages

The main advantage of per-bank refresh is that it parallelizes refreshes with read/write accesses to the same rank or channel. This is explained in Figure 73. Suppose, two read operations, one for bank 0 and one for bank 1 are waiting to get serviced. However, a pending all-bank request command is being sent prior to servicing the reads as shown in the figure on left. Since, all banks are blocked, both reads have to wait for the refresh operation to finish, only after which can they be serviced, incurring long memory-access latencies and DRAM bandwidth loss. In the per-bank refresh case (figure on right), when REF_{pb} for bank 0 is being serviced, read request of bank 1 can be sent to the DRAM. Similarly, read request of bank 0 can be sent in parallel with the refresh command for bank 1. Hence, in general, most read/write accesses remain unblocked while refreshing, incurring smaller memory-access latency and negligible DRAM bandwidth loss. We performed opportunistic per-bank refresh, i.e., anytime a bank is idle and the next refresh for that bank is possible in an $8 * t\_REFI$ window, we opportunistically performed per-bank refresh for that bank, allowing read/write accesses to other banks in parallel to the refresh operation.

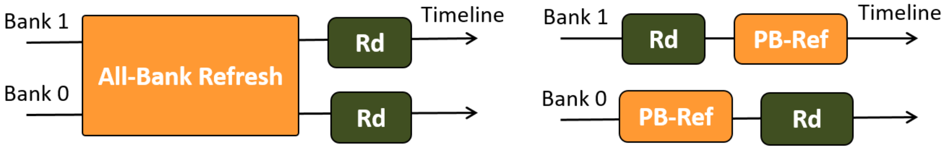


Figure 73: Comparison of all-bank and per-bank refresh management

The example above considers an ideal scenario where read accesses of other banks are always available during refresh. However, this scenario does not happen very often because of a few limitations and disadvantages of per-bank refresh, which will be discussed next.

8.5.2 Challenges

This section describes the challenges with per-bank refresh and discuss simple solutions that we used to solve these problems.

8.5.2.1 Bank round-robin order

The JEDEC standard defines per-bank refresh to be performed in a sequential round robin order, that is, an internal counter is maintained for the next bank to be refreshed, which is incremented every time a new refresh command is sent. This reduces the number of bits required for per-bank refresh command as the bank number to be refreshed does not have to be forwarded externally from the memory controller. However, this in-order refreshing policy limits the potential for improvement by not allowing refreshes to be performed in parallel with accesses. E.g., suppose we have a request stream accessing only one bank out of all the eight banks of the DRAM (Let's say bank 2). Suppose, refreshes are being done in a sequential round-robin order and suppose we have a policy which postpones refreshes because of active read/write requests. Thus, refresh for bank 2 gets postponed. The next refresh for bank 3 will get delayed as well (sequential nature dictates bank 3 cannot be served before bank 2) although there are no read/write request pending for bank 3. Similar is the case for refreshes for bank 4, bank 5, and so on. When eight such refreshes are aggregated, DRAM will stop servicing all requests and send all the refreshes in a burst, thus leading to DRAM bandwidth unavailability. It can be seen that in this scenario, no refresh will be performed in parallel with accesses, thus per-bank refreshing will have no advantage over the all-bank case. Furthermore, it will still incur the other disadvantages of per-bank refresh (explained later) making its performance worse than the all-bank case.

We solve the problem by allowing refreshes to be performed out-of-order as done in [13]. With out-of-order refreshing, pending refreshes for idle banks can get serviced while the refresh for a busy bank is waiting for read/write accesses to be finished. Hence, in the previous case, refresh for bank 3 will be serviced before refresh for bank 2, and so on; allowing read/write accesses to happen in parallel with refreshes.

8.5.2.2 *Increased t_RRD Penalty*

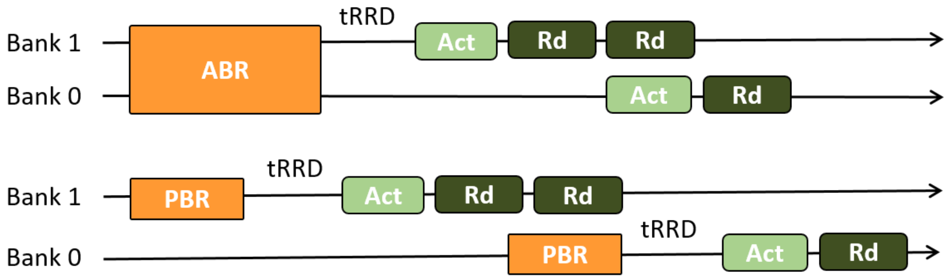


Figure 74: Increased t_RRD penalty

An important timing parameter which limits the performance of DRAMs is the row-row activation delay *t_RRD* between different banks of a DRAM. *t_RRD* is the minimum time between subsequent activate commands to different banks. It also specifies the time required before sending an activate command to any bank after sending a refresh command. In the case of per-bank refresh, this timing constraint occurs NUM_BANKS times more often than the case of all-bank refresh. The scenario is explained in Figure 74, in which the channel has to incur t_RRD penalty twice for a 2-bank system performing per-bank refresh as compared to only once in the all-bank refresh case. Not allowing a bank to activate can delay a read/write operation that could have been done in parallel with the refresh operations. Since, a 3D system has a large number of banks, this can limit the performance gains of per-bank refresh by almost 6%.

8.5.2.3 $t\_RFC_{pb} < t\_RFC_{ab} < NUM\_BANKS * t\_RFC_{pb}$

As mentioned before, the time to perform one REF_{pb} operation is smaller than the refreshing time to perform one REF_{ab} operation. However, the number of REF_{pb} operations are NUM_BANKS times more than REF_{ab} cases. The total time to perform at least one per-bank operation in all banks (i.e. $NUM\_BANKS * t\_RFC_{pb}$) is greater than t_RFC_{ab}. Hence, $t\_RFC_{pb} < t\_RFC_{ab} < NUM\_BANKS * t\_RFC_{pb}$. This phenomenon leads to the following problem with per-bank refresh.

8.5.2.4 Per-bank refresh problem with postponing

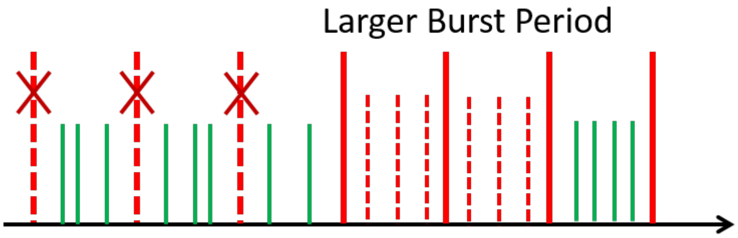
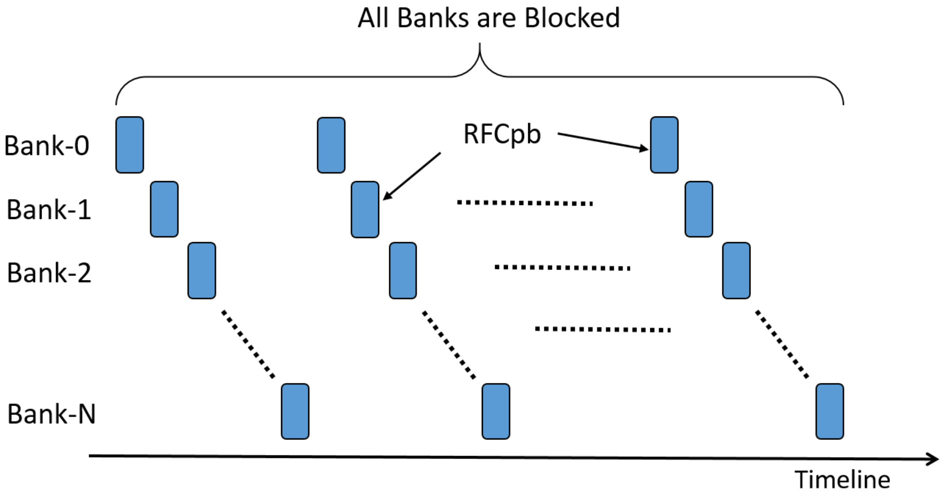
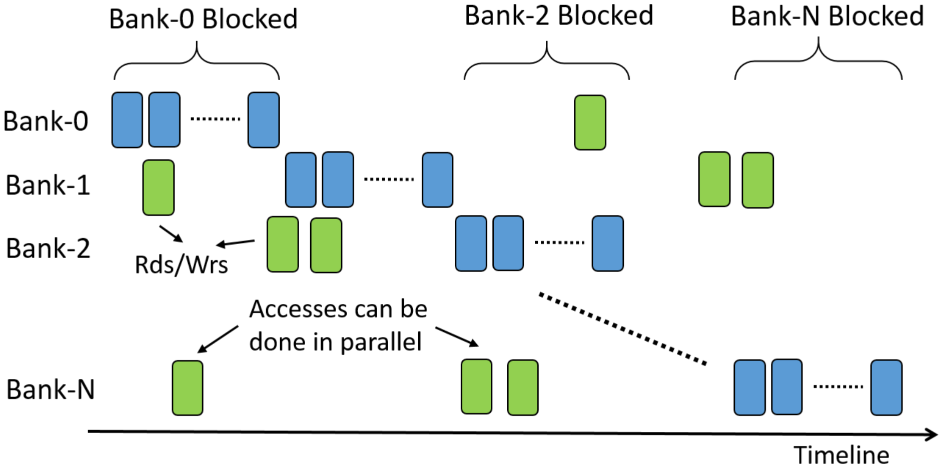


Figure 75: Per-bank postponed refresh

Figure 75 shows per-bank postponed refresh. Since the total time to perform eight per-bank refresh commands across all the banks, i.e. $8 * NUM\_BANKS * t\_RFC_{pb}$, is greater than performing eight all-bank commands, burst period in per-bank postponed refresh is larger than the burst-period in the all-bank postponed mode. Thus, all DRAM operations are being blocked for a much larger period in the per-bank postponed refresh case, making it significantly worse than its all-bank counterpart. This means that if DRAMs are operating in the postponed mode more often than the normal opportunistic mode, per-bank refresh can have larger performance penalty than the all-bank case. We solve this problem by parallelizing accesses with refreshes in the burst period as well; as explained in Figure 76a and 76b .



(a) Postponed Refresh - All banks scattered



(b) Postponed Refresh - One Bank at a time

Figure 76: Two competing postponed refresh options with per-bank refresh

Figure 76 indicates that refresh interleaving across banks in a burst period can be done in two different ways. The first case interleaves the refresh operations across banks, i.e. the first REF_{pb} refresh command is for bank 0, the next is for bank 1, and so on. This is shown in Figure 76a and termed as scattered per-bank refresh. The second case completes all refresh operations of bank 0 first, then send all refresh commands for bank 1, followed by refresh commands of bank 2, and so on. We called it optimized per-bank refresh. In the former case (scattered per-bank refresh), since refreshes to all the banks are being performed one after the other, it is difficult to maintain the strict timings of the burst mode and still allow parallel accesses. In the latter case (optimized per-bank refresh), access to other banks can be performed in parallel to refreshes while maintaining the strict timing requirements of the burst mode. The only operation not allowed is sending a read/write request to the next bank to be refreshed during the last refresh operation of the previous bank. Hence, the former case keeps all the banks blocked for the entire postponed period while the latter case only blocks one particular bank at a particular instant of time, thus allowing parallel accesses in the burst-refresh mode.

Another problem that occurs because of an elongated postponed period even in the second case is that if a read corresponding to the bank being refreshed gets stuck in the cache's MSHRs or come at the head of the ROB, no new accesses will arrive at the memory controller. Thus, there will be no access parallelization with refreshes limiting memory bandwidth. Another point to note is that since all-bank burst period is smaller, we can also switch to all-bank mode in the burst period (the approach used in [13]). However, this approach will not be possible in the case of variable rate refresh where different banks will be refreshing different row numbers at a particular time instant. Thus, in those scenarios, as will be explained later, switching from per-bank to all-bank mode will have high penalty. For our design, we stick to the per-bank refresh mode with postponing one bank at a time.

8.6 Variable-Rate Per-Bank Refresh Management (VPReM)

The previous section described the advantages and challenges of per-bank refresh and the approaches that we took to overcome these challenges. In this section, we discuss variable-rate refresh. We apply it with per-bank refresh to implement our variable-rate per-bank refresh (VPReM) scheme. We looked at various aspects of variations in temperature, such as temporal vs spatial variations, variations across different channels vs variations across different banks or dies, etc.

8.6.1 Fixed- vs Variable-Rate Refresh

In commodity DRAMs, refresh is performed at a fixed rate across all the channels and banks, typically at a period of 64ms for temperatures below $85°C$ and double that rate for every $10°C$ rise in temperature. Performing refresh at a fixed rate is feasible in 2D or 2.5D stacked memory systems, where the temperature variations across the DRAM dies and channels remain small. In such systems, the rise of temperature is dependent on the activity of different components of the DRAM itself, which are utilized rather uniformly across the dies. Furthermore, the overall DRAM temperature remains low. Thus, there is very low potential for exploiting temperature variations across the dies and channels. 3D systems, on the other hand, exhibits large variations across different dies and channels, as explained earlier. Since the frequency of refresh can be adjusted based on the operating temperature of the DRAM, variation in temperature can be exploited to perform refresh at a different rate for different channels and banks. We performed refresh for channels and banks at their own respective rate based on their operating temperature. We call this variable-rate refresh and used it for our VPReM policy.

A DRAM can exhibit large variations in temperature because of many reasons. In the next few subsections, we will briefly describe a few of those reasons.

8.6.2 Temporal vs Spatial Variations

As mentioned earlier, the temperature of a stack DRAM is dependent on the activity of the underlying multi-core die. This die exhibits large variations in temperature both temporally and spatially. Temporal variations occur because of various phases of the application, i.e. an application can have a varying range of activity in the cores at different instants of time running different phases of the program. For example, suppose a phase of a bigger program has many compute operations with a small memory footprint that can easily reside in the first-level cache. In this scenario, the activity of the cores and thus their temperature will be very high requiring high refresh rates. Now, suppose the program shifts to a phase where it has to access large amounts of data, which is placed highly irregularly in the memory and cannot be prefetched in the cache, such as pointer chasing accesses in heaps. In this scenario, the core will mostly be waiting for memory to fetch the data having very low activity, and thus the temperature will decrease. We detected the changes in temperature by using temperature sensors already provided in most DRAMs. We adjusted the refresh-rate of all the channels and banks based on this temperature according to Figure 69. We call this 'temperature-compensated refresh management' or TCReM. This is similar to temperature-compensated self-refresh (TCSR) but in the active DRAM mode. Furthermore, this requires temperature sensor information to be readable externally by the memory controller. TCReM can detect program phases and adjust the refresh rate accordingly saving refresh power. Furthermore, it can help improve performance since it reduces the refresh rate during high DRAM activity phases, thus decreases refresh performance penalty.

A problem with TCReM, however, is that it only exploits temporal variations in temperature, which occur only because of the changes in the program phases. As explained earlier, a multicore die exhibits high spatial variation in temperature as well, because of the varying activity of different cores specially in the case of heterogeneous systems. Another problem is that the rate of change in temperature is generally very slow, thus the potential for improvement with TCReM only is low. We can exploit spatial variations in temperature as well which is explained next.

8.6.3 Exploiting Variations Per Channel

Spatial variations in temperature of a 3D stack DRAM can be exploited at both per-channel or at a per-bank level. Different channels or vaults of a DRAM is stacked on different components (cores/caches/tiles) of the underlying multicore die. Each component of the underlying multicore die, such as a core or a cache, is running on its own thread with its own set of instructions and activity, thus has its own temperature. The difference in temperature of these components can be high as already discussed in section 8.3. Thus, the channels present on top of these components will have high variance in temperature. Each DRAM channel can be refreshed at its own rate based on its respective temperature. The memory controller for each channel can send refresh commands at their respective rate without any synchronization among different channels. In this case, we require one temperature sensor for each DRAM channel. However, since TCSR generally operates at a per-channel level, it is highly likely that modern DDRs have more than one sensors already, one per channel in this case. We call this scheme 'variable-rate per-channel refresh management' or VCReM.

VCReM has the advantage of exploiting spatial variations in temperature within a DRAM die. Thus, it allows some cores to operate at a much higher temperature or frequency and some to operate at a much lower frequency, but still keep the refresh power low. However, VCReM cannot exploit temperature differences across different dies. Thus, its potential for power saving is also limited.

8.6.4 Exploiting Variations Per Bank

Different dies of a 3D stacked DRAM operate at different temperature because of the uneven heat removal capability of the heat sinks present on top. This difference can be as large as $30°C$ in certain cases. We can exploit this difference in temperature by allowing different refresh rates for different sub-dies of a channel. Each sub-die consists of two or more banks for that channel. Allowing different rates for different dies means refreshing each bank of a DRAM at its own rate based on its own temperature. We call this scheme as 'variable-rate per-bank refresh management' or VPReM. This allows us to refresh each bank of a channel at a different rate, fully exploiting the variations in temperature of a 3D stacked DRAM. In this case, we require per-bank temperature sensors. We believe that a modern stacked DRAM already has temperature sensor for each sub-die of a channel. For this reason, we restrict changing the refresh rate to a granularity of a sub-die only, with all the banks in that sub-die to operate at the same rate. This will not reduce the advantages of VPReM, since the difference in temperature between different banks of a sub-die is generally very small and hence unlikely to require variable-rate refresh. VPReM requires DRAMs to operate in a per-bank mode since different refresh commands will be sent for each bank. Our VPReM policy combines the per-bank refresh optimizations discussed in section 8.5 with variable-rate refresh improving both performance and power of a 3D stacked memory and allow it to tolerate hotspots in the memory.

8.7 Results

Next, we will discuss some power and performance results of our variable-rate per-bank refresh management scheme but first we will describe our evaluation methodology for this study.

8.7.1 Evaluation Methodology

We evaluated our refresh management schemes using the same simulation infrastructure as described in section 7.1. The baseline 3D system compose of a 4x4 tiled architecture with 32 simple out-of-order cores and sixteen memory channels as discussed in section 3.2. Any 2.5D design used for comparison purposes consists of 4 channels as described in section 7.1. Other configurations, such as the core, cache, and DRAM configurations remain the same. As mentioned before, manifold [112], kitfox [90], and usimm [14] is used to estimate power of different dies, which is fed back to 3D-ICE [92] thermal model to calculate temperature of different channels and banks. At every time epoch, the calculated steady-state temperatures are then used to calculate refresh rates of different dies and channels, which is fed back to usimm to calculate the updated performance and power results.

8.7.2 Per-Bank Refresh Management Results

We first compare the performance of various per-bank refresh management schemes with the all-bank refresh case. Figure 77 gives the performance of various benchmarks with different refresh mechanisms normalized to the case where refresh has been turned off $no-refresh$. Note that the refresh rate is set to be 16ms, a high operating temperature scenario. In the figure, oab means an opportunistic all-bank refresh scheme while opb means an opportunistic per-bank refresh. $opb-S$ represents scattered per-bank refresh while $opb-O$ represents our optimized approach. ppb represents the case where the opportunistic feature of per-bank refresh is turned off and all refreshes happen in a postponed mode. We only present the second case of one bank at a time in the postponed burst period.

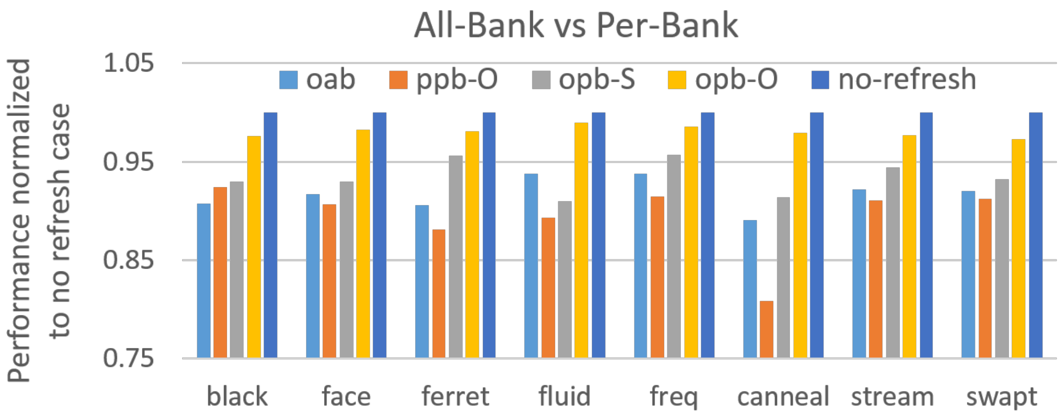
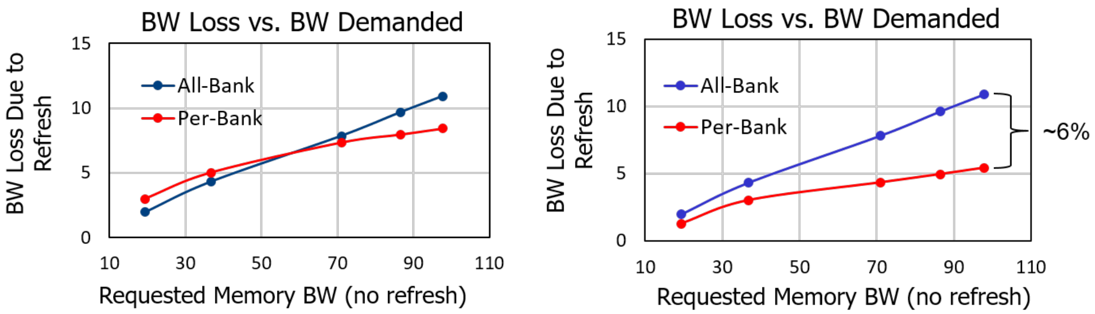


Figure 77: Performance of various per-bank refresh management schemes normalized to the no-refresh case

The results show that the all-bank refresh case exhibit almost 9% performance loss on average compared to the case with $no-refresh$. Note that we have used opportunistic all-bank refresh, where the refresh operation can be done opportunistically if all the banks of a channel are empty. However, for high refresh rates, this scenario does not occur too often incurring performance loss. The opportunistic per-bank cases, on the other hand, performs better than the all-bank case for all the benchmarks. This indicates that per-bank refresh is able to find opportunities to refresh banks while accesses to other banks are being serviced. The performance decrease with $opb - S$ as compared to the $no - refresh$ case is around 6.5% on average. However, the difference reduces to less than 2% with our optimized approach in the postponed burst mode. To understand this difference, recall that $opb - S$ does not allow parallel accesses with refreshes in the postponed burst mode while $opb - O$ does allow them. Also, recall that the total time taken by per-bank refresh in the postponed mode is much larger than the postponed mode of the all-bank case, and hence occurs more frequently. This is indicated by the case with postponed per-bank refresh ppb, which has the least performance for most of the benchmarks, an average performance loss of 10.5% as compared to the $no - refresh$ case. The difference in performance between the two opportunistic per-bank approaches means that a significant number of refresh operations does not find opportunities to be performed in parallel with read/write accesses in the normal mode due to various timing constraints of the DRAM, and have to resort to the postponed burst mode in order to get serviced. Hence, our optimized approach which let's accesses to be performed along with refreshes in the burst mode significantly improves performance, with IPC reaching almost close to ideal, even with a significantly higher refresh rate of 16ms. We conclude by saying that our optimized per-bank scheme can be used to perform refreshes at significantly higher rates with negligible performance loss. We can utilize this fact to increase the thermal operating range of 3D stacks, allowing them to operate at much higher temperature and thus frequency, improving the performance of the overall system.

Figure 78 compares the bandwidth loss observed with all-bank and per-bank refresh for the canneal benchmark from a scenario that has no refresh. The increase in requested bandwidth is achieved by pumping up the frequency of the cores. Figure 78a compares scattered per-bank refresh $opb - S$ with all-bank refresh while Figure 78b compares our optimized per-bank approach $opb - O$. In both the cases, increasing bandwidth demand



(a) Bandwidth Loss due to Refresh - Scattered Mode

(b) BW Loss due to Refresh - Optimized Mode

Figure 78: Bandwidth loss due to refresh with various refresh management schemes

increases the performance loss due to refresh as more and more read/write operations gets in conflict with refresh operations. Canneal is a memory intensive application, and hence operates in the postponed burst mode often. For $opb - S$, its bandwidth loss and hence performance decrease by operating in the postponed mode is higher than the all-bank case at lower frequencies. With increasing bandwidth demand, the rate of increase of bandwidth loss reduces since the per-bank feature finds more refreshes in parallel with read/write accesses. On the other hand, for the all-bank case, the bandwidth loss keeps increasing linearly reaching more than 11% for high bandwidth demand. The optimized per-bank refresh, as shown in Figure 78b, however outperforms all-bank refresh in all scenarios because of its ability to perform accesses in parallel to refreshes even in the postponed burst mode. The bandwidth difference between the all-bank refresh and the optimized per-bank refresh is greater than 6%.

8.7.3 VPReM Results

This section discusses the results of our variable-rate refresh management (VPReM) scheme. Figure 79 gives the performance overhead of various refresh management options by changing the number of cores. Performance overhead is defined as the percentage difference in cycles with a case in which refresh has been turned off. Here, $ab - opr$ represents opportunistic all-bank refresh while $pb - opr$ indicates our optimized opportunistic per-bank technique. Both implements fixed-rate refresh at a period of 16ms. We chose 16ms for our comparison purposes because it corresponds to peak temperature of the whole program. *var* indicates VPReM, the variable-rate per-bank refresh management scheme. We also implemented a per-bank refresh mechanism in which at each epoch, the peak temperature of all the banks is calculated and used to set the refresh rate of all channels and banks of the DRAM. This scheme captures temporal variations but loses any spatial variations in the DRAM, similar to our TCReM approach and indicated by $pb - peak$.

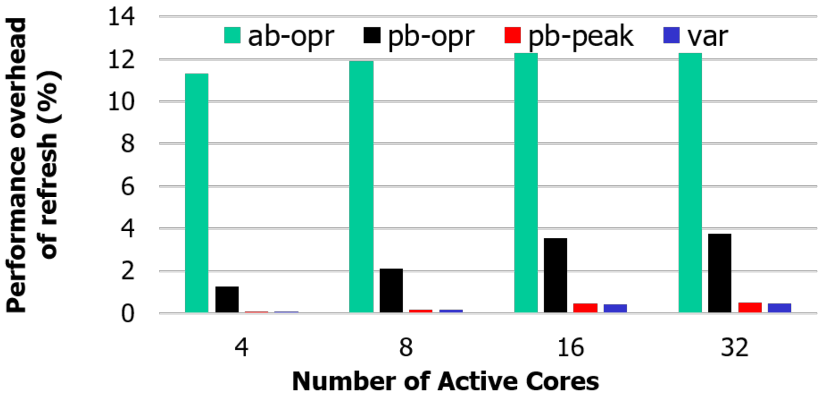
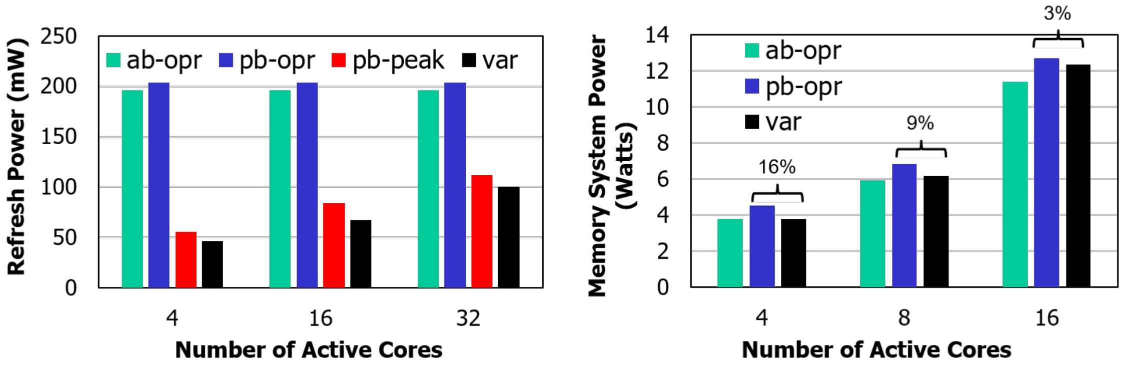


Figure 79: Performance overhead with different variable-rate refresh schemes

$ab - opr$ has the lowest performance because of no parallelization of refreshes and accesses as indicated in section 8.7.2. $pb - opr$, on the other hand, has a much smaller overhead (around 1-4%) as explained earlier. The performance overhead for both the cases increases with the increase in the number of cores and thus higher bandwidth requirement. However, with variable-rate refresh (both $pb - peak$ and *var*), the performance penalty is negligible (<1%). This is because both TCReM and VPReM captures the variations in temperature of a DRAM well and refreshes the DRAM accordingly. Thus, for program phases where the compute intensity and thus temperature is low, temperature compensation of our schemes allow us to operate at very low refresh rates reducing performance penalty. However, when the intensity is high, we can bump up the refresh rate and still incur low performance overhead with the optimized per-bank approach.

Figure 80a gives the power advantages of variable-rate refresh. Both $ab - opr$ and



(a) Refresh Power Savings

(b) Memory Power Savings

Figure 80: Power savings with various variable-rate refresh management schemes

$pb - opr$ has high refresh power because of their very high refresh rate. $pb - opr$ shows a slightly higher value (almost 4% greater) because it consumes the same amount of energy in a shorter span of time (superior performance), thus increasing dynamic power. $pb - peak$ and *var*, which represents TCReM and VPReM, can adjust to different temperature phases. Thus, their power is much lower than the fixed-rate schemes. The refresh power improvement of $pb - peak$ and *var* are 57% and 63% on average, respectively from the $ab - opr$ case; indicating that temperature of an application changes with time and a temperature-compensated refresh mechanism can reduce the negative impacts associated with refreshes, significantly. The difference between $pb - peak$ and *var*, however, is small (almost 16% power improvement). The results show that our simulation mechanism is able to capture temporal variations well. However, spatial variations and their advantages are low. We believe that this is because of the homogeneous cores and multiple instances of the same application running across different cores, making the power profile across cores and thus DRAM channels very similar. We will extend our simulations and use heterogeneous cores and benchmarks in future to exploit the advantages of spatial variations as well.

Figure 80b gives the overall memory power advantages. Again, $ab - opr$ has the least power because of its higher number of cycles, and $pb - opr$ has the highest power because of its high refresh rate. VPREM or *var* has lesser power than the $pb - opr$ case (average improvement of 9.7%), although it has higher performance and hence lesser number of cycles. The power advantage of VPReM, however, reduces with increasing core count, e.g., from 16% in 4-core case to 3% in a system with 16 active cores. More cores increases the temperature of DRAMs and hence increases its refresh rate, thus the potential for power saving using variable-rate refresh becomes low.

8.8 Concluding Remarks

We conclude the chapter by restating that refresh power is a bigger challenge in 3D stacked DRAMs than conventional 2D memories specially in active DRAM modes. In such a case, refreshing all the channels and banks of the stack based on the hottest operating temperature increases total execution time and energy degrading both performance and power. VPReM allows us to exploit the variation in 3D DRAM temperature and distributes the refresh power non-uniformly across the stack allowing the DRAM to operate at much higher temperatures tolerating hot-spots in the memory without incurring performance loss. It can further be used to bump up the frequency or activity of certain regions while keeping the temperature of other regions low, thus improving performance. The per-bank feature of VPReM reduces the performance penalty of operating at higher temperatures while the variable-rate feature allows us to adapt to spatial and temporal variations. Our results indicate improvements in both performance and power with significant power savings because of temporal variations. Exploiting spatial variations to reduce power by exploring more heterogeneous domains will be done in future.

CHAPTER 9
CONCLUSION

This dissertation looked at the advantages and challenges of high parallelism in 3D stacked memory systems. Parallelism, in such systems, alters the fundamental relationships between latency, bandwidth, and energy of the memory hierarchy. We re-evaluated these relationships and identified three key challenges; network latency, parallelism management using address translations, and thermal regulation; that were addressed in this thesis. We first established the need for fine-grained, highly-parallel 3D memory systems and characterized their impact on latency in various components of the cache and the memory subsystem. We also established the role of hardware address translations in regulating locality vs parallelism and bandwidth vs power trade-off in concurrent memory channels. Keeping these observations in mind, we proposed a memory subsystem re-organization of the baseline system that places L2 cache banks next to DRAMs with an interconnection network only between the L1 and the L2, thus reducing the traffic in the network. This re-organization is achieved by coupling the addressing scheme of a distributed last-level cache (LLC) with that of a distributed high-channel-count main memory.

Next, to reduce the network latency further, we designed a single-cycle *centralized-buffer router* (CBR) [33] that supports high-radix networks with small dependence of buffer area on radix. We reduced the buffer space of centralized buffer routers by proposing the use of variants of bubble flow control; specially for high-radix networks. We explored both deterministic and adaptive versions of CBR and analyzed their performance and power advantages with both regular and irregular topologies under different traffic scenarios. Such low-cost, low-latency routers are ideally suited for a 3D stacked system with large number of memory channels.

Lastly, we analyzed 3D stack's thermal problem and characterized its impact on system performance and reliability. We identified the rise in temperature and its variance within and across the dies as the two major thermal bottlenecks leading to increased DRAM leakage and hence higher refresh rate. We showed that newer cooling solutions, such as microfluidics cooling, can help alleviate 3D stacks temperature problem and allow it to run new compute-intensive applications and operate at much higher frequencies utilizing the full potential of the high bandwidth of 3D memories. To exploit the variance in stack DRAM's temperature, we proposed a variable-rate per-bank refresh management scheme that not only reduces 3D stacks refresh power but also increases its operating range by allowing DRAMs to function correctly at significantly higher temperatures.

In short, this dissertation proposed various modifications and optimizations to an exemplar 3D system that exploits its high concurrency to reduce the power and performance bottlenecks of such a system. All optimizations proposed requires small changes in the existing architecture and can be adopted easily in the near future. Future work includes using address management to regulate bandwidth vs. power trade-off of multiple channels, more tight coupling of last-level cache and memory controller policies in 3D memory systems, providing quality of service support with centralized buffer routers, and exploring solutions for geometry challenges of electrical TSVs and thermal pin-fins with the microfluidics cooling technology.

REFERENCES

[1] Ahn, J. H., Jouppi, N. P., Kozyrakis, C., Leverich, J., and Schreiber, R. S., "Future scaling of processor-memory interfaces," in *Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pp. 42:1–42:12, 2009.

[2] Allan, G., "Ddr4 bank groups in embedded applications," May 2013.

[3] Awasthi, M., Nellans, D. W., Sudan, K., Balasubramonian, R., and Davis, A., "Handling the problems and opportunities posed by multiple on-chip memory controllers," in *Parallel architectures and compilation techniques*, PACT '10, pp. 319–330, 2010.

[4] Besta, M., Hassan, S., Hoefler, T., and Yalamanchili, S., "A Family of Energy-Efficient Low-Diameter On-Chip Networks for Thousand Core Chips (Submitted)," in *International Symposium on High-Performance Computer Architecture*, March 2016.

[5] Besta, M. and Hoefler, T., "Slim Fly: A Cost Effective Low-Diameter Network Topology," in *High Performance Computing, Networking, Storage and Analysis*, Nov. 2014.

[6] Beu, J. G., Rosier, M. C., and Conte, T. M., "Manager-client pairing: a framework for implementing coherence hierarchies," MICRO-44, 2011.

[7] Beyne, E., "3d system integration technologies," in *VLSI Technology, Systems, and Applications, 2006 International Symposium on*, pp. 1 –9, april 2006.

[8] Bhati, I., Chang, M.-T., Chishti, Z., Lu, S.-L., and Jacob, B., "Dram refresh mechanisms, penalties, and trade-offs," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 108–121, 2016.

[9] Bhati, I., Chishti, Z., Lu, S.-L., and Jacob, B., "Flexible auto-refresh: enabling scalable and energy-efficient dram refresh reductions," in *ACM SIGARCH Computer Architecture News*, vol. 43, pp. 235–246, ACM, 2015.

[10] Bienia, C., Kumar, S., Singh, J. P., and Li, K., "The parsec benchmark suite: Characterization and architectural implications," tech. rep., IN PRINCETON UNIVERSITY, 2008.

[11] Canwen, X., Minxuan, Z., Yong, D., and Zhitong, Z., "Dimensional bubble flow control and fully adaptive routing in the 2-d mesh network on chip," in *Embedded and Ubiquitous Computing, 2008.*, dec. 2008.

[12] Chandrasekar, K., *High-level power estimation and optimization of DRAMs*. PhD thesis, TU Delft, Delft University of Technology, 2014.

[13] Chang, K. K.-W., Lee, D., Chishti, Z., Alameldeen, A. R., Wilkerson, C., Kim, Y., and Mutlu, O., "Improving dram performance by parallelizing refreshes with accesses," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 356–367, IEEE, 2014.

[14] Chatterjee, N., Balasubramonian, R., Shevgoor, M., Pugsley, S., Udipi, A., Shafiee, A., Sudan, K., Awasthi, M., and Chishti, Z., "Usimm: the utah simulated memory module," *University of Utah, Tech. Rep*, 2012.

[15] Chen, C.-H. O., Park, S., Krishna, T., Subramanian, S., Chandrakasan, A. P., and Peh, L.-S., "Smart: A single-cycle reconfigurable noc for soc applications," in *Conference on Design, Automation and Test in Europe*, DATE '13, pp. 338–343, 2013.

[16] Chen, L. and Pinkston, T. M., "Worm-bubble flow control," in *High Performance Computer Architecture (HPCA), 2013*.

[17] Chen, L., Wang, R., and Pinkston, T., "Critical bubble scheme: An efficient implementation of globally aware network flow control," in *IPDPS, 2011*.

[18] Cong, J., Wei, J., and Zhang, Y., "A thermal-driven floorplanning algorithm for 3d ics," in *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pp. 306–313, IEEE, 2004.

[19] Dally, W. and Towles, B., *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[20] Das, R., Mutlu, O., Moscibroda, T., and Das, C. R., "Aérgia: exploiting packet latency slack in on-chip networks," in *ISCA '10: 37th annual international symposium on Computer architecture*, (New York, NY, USA), pp. 106–116, ACM, 2010.

[21] Derner, S. J., Kurth, C. R., and Habersetzer, D. L., "Partial array self-refresh," Dec. 21 2004. US Patent 6,834,022.

[22] Duato, J., Yalamanchili, S., and Ni, L., *Interconnection networks*. Morgan Kaufmann Publishers Inc., 2002.

[23] Egawa, R., Taday, J., Kobayashi, H., and Gotoy, G., "Evaluation of fine grain 3-d integrated arithmetic units," in *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*, pp. 1 –8, sept. 2009.

[24] et al, D. H. K., "3d-maps: 3d massively parallel processor with stacked memory," in *Solid-State Circuits Conference (ISSCC), 2012 IEEE International*, feb. 2012.

[25] Ganesh, B., Jaleel, A., Wang, D., and Jacob, B., "Fully-buffered dimm memory architectures: Understanding mechanisms, overheads and scaling," in *High Performance Computer Architecture, 2007. HPCA 2007.*, pp. 109–120, 2007.

[26] Grot, B., Hestness, J., Keckler, S., and Mutlu, O., "Kilo-noc: A heterogeneous network-on-chip architecture for scalability and service guarantees," in *ISCA, 2011*.

[27] Grot, B., Hestness, J., Keckler, S., and Mutlu, O., "Express Cube Topologies for on-Chip Interconnects," in *HPCA 2009.*, pp. 163–174, Feb 2009.

[28] Guan, M. and Wang, L., "Temperature aware refresh for dram performance improvement in 3d ics," in *Sixteenth International Symposium on Quality Electronic Design*, pp. 207–211, IEEE, 2015.

[29] Hassan, S., Choudhary, D., Rasquinha, M., and Yalamanchili, S., "Regulating locality vs. parallelism tradeoffs in multiple memory controller environments," in *Parallel Architectures and Compilation Techniques (PACT)*, pp. 187 –188, oct. 2011.

[30] Hassan, S. and Yalamanchili, S., "Near Data Processing: Impact and Optimization of 3D Memory System Architecture on the Uncore," in *The International Symposium on Memory Systems*, Oct. 2015.

[31] Hassan, S. M. and Yalamanchili, S., "Bubble sharing: Area and energy efficient adaptive routers using centralized buffers," in *NoCS 2014*.

[32] Hassan, S. M. and Yalamanchili, S., "Centralized buffer router: A low latency, low power router for high radix nocs," in *NOCS*, 2013.

[33] Hassan, S. M. and Yalamanchili, S., "Centralized buffer router with elastic links and bubble flow control," in *Tech. Report, git-cercs-13-03*, 2013.

[34] Ho Song, Y. and Pinkston, T. M., "A progressive approach to handling message-dependent deadlock in parallel computer systems," *Parallel Distrib. Syst.*, 2003.

[35] Hur, I. and Lin, C., "A comprehensive approach to dram power management," in *HPCA-14*, pp. 305–316, Feb 2008.

[36] Jacob, B., Ng, S., and Wang, D., *Memory Systems: Cache, DRAM, Disk.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[37] Jasperson, B. A., Jeon, Y., Turner, K. T., Pfefferkorn, F. E., and Qu, W., "Comparison of micro-pin-fin and microchannel heat sinks considering thermal-hydraulic performance and manufacturability," *Components and Packaging Technologies, IEEE Transactions on*, vol. 33, no. 1, pp. 148–160, 2010.

[38] Jerger, N. E., Liu, Z., and Wang, Z., "Leaving one slot empty: Flit bubble flow control for torus cache-coherent nocs," *IEEE Transactions on Computers*, 2013.

[39] J.Lira, C. and A.Gonzlez., "Analysis of non-uniform cache architecture policies for chip-multiprocessors using the parsec benchmark suite," in *In Proceedings of the 2nd Workshop on Managed Many-Core Systems*, MMCS'09, (Washington D.C, (USA)), in conjunction with the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09), March 2009.

[40] Kaseridis, D., Stuecheli, J., and John, L. K., "Minimalist open-page: a dram page-mode scheduling policy for the many-core era," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44 '11, (New York, NY, USA), pp. 24–35, ACM, 2011.

[41] Keeton, K., "The machine: An architecture for memory-centric computing," in *Workshop on Runtime and Operating Systems for Supercomputers*, ROSS, 2015.

[42] Kersey, Chad, *QSim - QEMU-based Emulation Library for Microarchitecture Simulation*.

[43] Kim, J., "Low-cost router microarchitecture for on-chip networks," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 255–266, ACM, 2009.

[44] Kim, J., Dally, W. J., Scott, S., and Abts, D., "Technology-Driven, Highly-Scalable Dragonfly Topology," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, (Washington, DC, USA), pp. 77–88, IEEE Computer Society, 2008.

[45] Kim, J., Dally, W. J., and Abts, D., "Flattened Butterfly: A Cost-efficient Topology for High-radix Networks," ISCA '07, 2007.

[46] Kim, Y., Han, D., Mutlu, O., and Harchol-Balter, M., "Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *High Performance Computer Architecture (HPCA), 2010*, pp. 1 –12, 9-14 2010.

[47] Kim, Y., Seshadri, V., Lee, D., Liu, J., and Mutlu, O., "A case for exploiting subarray-level parallelism (salp) in dram," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pp. 368–379, IEEE, 2012.

[48] Kumar, A., Peh, L.-S., Kundu, P., and Jha, N., "Toward ideal on-chip communication using express virtual channels," *Micro, IEEE*, jan.-feb. 2008.

[49] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P., "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, (New York, NY, USA), pp. 469–480, ACM, 2009.

[50] Liu, C., Ganusov, I., Burtscher, M., and Tiwari, S., "Bridging the processor-memory performance gap with 3d ic technology," *Design Test of Computers, IEEE*, vol. 22, pp. 556 – 564, nov.-dec. 2005.

[51] Liu, J., Jaiyen, B., Veras, R., and Mutlu, O., "Raidr: Retention-aware intelligent dram refresh," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pp. 1–12, IEEE, 2012.

[52] Liu, S., Pattabiraman, K., Moscibroda, T., and Zorn, B. G., “Flikker: Saving dram refresh-power through critical data partitioning,” *SIGPLAN Not.*, 2011.

[53] Loh, G. H., “3d-stacked memory architectures for multi-core processors,” *SIGARCH Comput. Archit. News*, vol. 36, pp. 453–464, June 2008.

[54] Loh, G. H., “3d-stacked memory architectures for multi-core processors,” in *ACM SIGARCH computer architecture news*, vol. 36, pp. 453–464, IEEE Computer Society, 2008.

[55] Ma, S., Jerger, N. E., and Wang, Z., “Whole packet forwarding: Efficient design of fully adaptive routing algorithms for networks-on-chip,” in *HPCA*, 2012.

[56] Matsutani, H., Koibuchi, M., Amano, H., and Yoshinaga, T., “Prediction router: Yet another low latency on-chip router architecture,” in *HPCA 2009*.

[57] McKay, B. D., Miller, M., and Širáň, J., “A note on large graphs of diameter two and given maximum degree,” *Journal of Combinatorial Theory, Series B*, vol. 74, no. 1, pp. 110 – 118, 1998.

[58] Michelogiannakis, G., Balfour, J., and Dally, W., “Elastic-buffer flow control for on-chip networks,” in *High Performance Computer Architecture, HPCA 2009*.

[59] Michelogiannakis, G. and Dally, W., “Elastic buffer flow control for on-chip networks,” *Computers, IEEE Transactions on*, 2011.

[60] Micron Technology, *MT41J128M8BY-18E*.

[61] Moscibroda, T. and Mutlu, O., “A case for bufferless routing in on-chip networks,” in *ACM SIGARCH Computer Architecture News*, 2009.

[62] Motoyoshi, M., “Through-silicon via (tsv),” *Proceedings of the IEEE*, vol. 97, pp. 43–48, Jan 2009.

[63] Mukundan, J., Hunter, H., Kim, K.-h., Stuecheli, J., and Martínez, J. F., “Understanding and mitigating refresh overheads in high-density ddr4 dram systems,” in *ACM SIGARCH Computer Architecture News*, vol. 41, pp. 48–59, ACM, 2013.

[64] Muralidhara, S. P., Subramanian, L., Mutlu, O., Kandemir, M., and Moscibroda, T., “Reducing memory interference in multicore systems via application-aware memory channel partitioning,” MICRO-44, pp. 374–385, 2011.

[65] Mutlu, O. and Moscibroda, T., “Parallelism-aware batch scheduling: Enabling high-performance and fair shared memory controllers,” *Micro, IEEE*, pp. 22 –32, 2009.

[66] Nair, P., Chou, C.-C., and Qureshi, M. K., “A case for refresh pausing in dram memory systems,” in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 627–638, IEEE, 2013.

[67] Pawlowski, J., "Hybrid memory cube: Breakthrough dram performance with a fundamentally re-architected dram subsystem," in *Hot Chips*, 2011.

[68] Peh, L.-S. and Dally, W., "A delay model and speculative architecture for pipelined routers," in *HPCA 2001.*

[69] Puente, V., Beivide, R., Gregorio, J., Prellezo, J., Duato, J., and Izu, C., "Adaptive bubble router: a design to improve performance in torus networks," in *ICPP, 1999.*

[70] Puente, V., Izu, C., Beivide, R., Gregorio, J., Vallejo, F., and Prellezo, J. M., "The adaptive bubble router," 2001.

[71] Pugsley, S., Jestes, J., Balasubramonian, R., Srinivasan, V., Buyuktosunoglu, A., Davis, A., and Li, F., "Comparing implementations of near-data computing with in-memory mapreduce workloads," *Micro, IEEE*, vol. 34, pp. 44–52, July 2014.

[72] Qureshi, M. K., Kim, D.-H., Khan, S., Nair, P. J., and Mutlu, O., "Avatar: A variable-retention-time (vrt) aware refresh for dram systems," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 427–437, IEEE, 2015.

[73] Qureshi, M. K. and Loh, G. H., "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *International Symposium on Microarchitecture*, pp. 235–246, 2012.

[74] Rahmati, D., Sarbazi-Azad, H., Hessabi, S., and Kiasari, A. E., "Power-efficient deterministic and adaptive routing in torus networks-on-chip," *Microprocessors and Microsystems*, 2012.

[75] Ramanujam, R., Soteriou, V., Lin, B., and Peh, L.-S., "Design of a high-throughput distributed shared-buffer noc router," in *Networks-on-Chip (NOCS), 2010.*

[76] Rasquinha, M., Hassan, S. M., Song, W., Chae, K., Cho, M., Mukhopadhyay, S., and Yalamanchili, S., "System impact of 3d processor-memory interconnect: A limit study," in *Tech. Report, git-cercs-11-04*, 2011.

[77] Restle, P. J., Park, J., and Lloyd, B. F., "Dram variable retention time," in *Electron Devices Meeting, 1992. IEDM'92. Technical Digest., International*, pp. 807–810, IEEE, 1992.

[78] Rixner, S., Dally, W. J., Kapasi, U. J., Mattson, P., and Owens, J. D., "Memory access scheduling," in *ISCA '00*, (NY, USA), pp. 128–138, ACM, 2000.

[79] Rogers, B. M., Krishna, A., Bell, G. B., Vu, K., Jiang, X., and Solihin, Y., "Scaling the bandwidth wall: challenges in and avenues for cmp scaling," in *36th annual international symposium on Computer architecture*, ISCA '09, pp. 371–382, 2009.

[80] Ros, A., Acacio, M., and Garcia, J., "Dico-cmp: Efficient cache coherency in tiled cmp architectures," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1 –11, april 2008.

[81] Rosenfeld, P., Cooper-Balis, E., and Jacob, B., "Dramsim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, vol. 10, pp. 16–19, Jan. 2011.

[82] Sadri, M., Jung, M., Weis, C., Wehn, N., and Benini, L., "Energy optimization in 3d mpsocs with wide-i/o dram using temperature variation aware bank-wise refresh," in *Proceedings of the conference on Design, Automation & Test in Europe*, p. 281, European Design and Automation Association, 2014.

[83] Sarvey, T. E., Zhang, Y., Zheng, L., Thadesar, P., Gutala, R., Cheung, C., Rahman, A., and Bakir, M. S., "Embedded cooling technologies for densely integrated electronic systems," in *Custom Integrated Circuits Conference (CICC), 2015 IEEE*, pp. 1–8, IEEE, 2015.

[84] Seitanidis, I., Psarras, A., Dimitrakopoulos, G., and Nicopoulos, C., "ElastiStore: An Elastic Buffer Architecture for Network-on-chip Routers," in *Conference on Design, Automation & Test in Europe*, DATE '14, pp. 240:1–240:6, 2014.

[85] Seongil, O., Choo, S., and Ahn, J. H., "Exploring energy-efficient dram array organizations," in *Circuits and Systems (MWSCAS), 2011*, pp. 1 –4, aug. 2011.

[86] Shi, B., Srivastava, A., and Bar-Cohen, A., "Hybrid 3d-ic cooling system using micro-fluidic cooling and thermal tsvs," in *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*, pp. 33–38, IEEE, 2012.

[87] shiuan Peh, L. and Dally, W. J., "Flit-reservation flow control," *IEEE Transactions on Parallel and Distributed Systems*, 2000.

[88] Sim, J., Alameldeen, A. R., Chishti, Z., Wilkerson, C., and Kim, H., "Transparent hardware management of stacked dram as part of memory," in *Microarchitecture (MICRO), 47th Annual IEEE/ACM International Symposium on*, pp. 13–24, 2014.

[89] Son, Y. H., Seongil, O., Yang, H., Jung, D., Ahn, J. H., Kim, J., Kim, J., and Lee, J. W., "Microbank: Architecting through-silicon interposer-based main memory systems," in *High Performance Computing, Networking, Storage and Analysis*, SC '14, 2014.

[90] Song, W. J., Mukhopadhyay, S., and Yalamanchili, S., "Kitfox: Multiphysics libraries for integrated power, thermal, and reliability simulations of multicore microarchitecture," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 5, no. 11, pp. 1590–1601, 2015.

[91] Song, W. J., Mukhopadhyay, S., and Yalamanchili, S., "Managing performance-reliability tradeoffs in multicore processors," in *Reliability Physics Symposium (IRPS), 2015 IEEE International*, pp. 3C–1, IEEE, 2015.

[92] Sridhar, A., Vincenzi, A., Ruggiero, M., Brunschwiler, T., and Atienza, D., "3d-ice: Fast compact transient thermal modeling for 3d ics with inter-tier liquid cooling," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 463–470, IEEE Press, 2010.

[93] Standard, J., "Wide i/o single data rate (wide i/o sdr) (jesd229)," Dec. 2011.

[94] Standard, J., "Ddr4 sdram standard (ddr4) (jesd79-4a)," Nov. 2013.

[95] Standard, J., "High bandwidth memory (hbm) dram (jesd235)," Oct. 2013.

[96] Standard, J., "Low power double data rate 4 (lpddr4) (jesd209-4)," August 2014.

[97] Standard, J., "Wide i/o 2 (wideio2) (jesd229-2)," August 2014.

[98] Stuecheli, J., Kaseridis, D., Hunter, H. C., and John, L. K., "Elastic refresh: Techniques to mitigate refresh penalties in high density memory," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 375–384, IEEE, 2010.

[99] Sudan, K., Chatterjee, N., Nellans, D., Awasthi, M., Balasubramonian, R., and Davis, A., "Micro-pages: increasing dram efficiency with locality-aware data placement," *SIGARCH Comput. Archit. News*, vol. 38, no. 1, pp. 219–230, 2010.

[100] Sun, H., Liu, J., Anigundi, R., Zheng, N., Lu, J.-Q., Rose, K., and Zhang, T., "3d dram design and application to 3d multicore systems," *Design Test of Computers, IEEE*, vol. 26, pp. 36 –47, sept.-oct. 2009.

[101] Tamir, Y. and Frazier, G., "Dynamically-allocated multi-queue buffers for vlsi communication switches," *Computers, IEEE Transactions on*, 1992.

[102] Technology, M., "Uprating semiconductors for high-temperature applications," pp. pp. 1–14, 2004.

[103] Technology, M., "Mobile dram power-saving features and power calculations," pp. pp. 1–10, 2009.

[104] Tran, A. T. and Baas, B. M., "Roshaq: High-performance on-chip router with shared queues," in *ICCD 2011*.

[105] Tsai, Y.-F., Wang, F., Xie, Y., Vijaykrishnan, N., and Irwin, M., "Design space exploration for 3-d cache," *Very Large Scale Integration (VLSI) Systems*, april 2008.

[106] Udipi, A. N., Muralimanohar, N., Chatterjee, N., Balasubramonian, R., Davis, A., and Jouppi, N. P., "Rethinking dram design and organization for energy-constrained multi-cores," *ACM SIGARCH Computer Architecture News*, vol. 38, pp. 175–186, 2010.

[107] Venkatesan, R. K., Herr, S., and Rotenberg, E., "Retention-aware placement in dram (rapid): Software methods for quasi-non-volatile dram," in *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.*, pp. 155–165, IEEE, 2006.

[108] Walton, M., "Hbm explained: Can stacked memory give amd the edge it needs?," May 2015.

[109] Wan, Z., Kim, Y. J., and Joshi, Y. K., "Compact modeling of 3d stacked die inter-tier microfluidic cooling under non-uniform heat flux," in *ASME 2012 International Mechanical Engineering Congress and Exposition*, pp. 911–917, American Society of Mechanical Engineers, 2012.

[110] Wan, Z., Xiao, H., Joshi, Y., and Yalamanchili, S., "Co-design of multicore architectures and microfluidic cooling for 3d stacked ics," *Microelectronics Journal*, 2014.

[111] Wang, H.-S., Zhu, X., Peh, L.-S., and Malik, S., "Orion: a power-performance simulator for interconnection networks," in *MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, (Los Alamitos, CA, USA), pp. 294–305, IEEE Computer Society Press, 2002.

[112] Wang, J., Beu, J., Bheda, R., Conte, T., Dong, Z., Kersey, C., Rasquinha, M., Riley, G., Song, W., Xiao, H., Xu, P., and Yalamanchili, S., "Manifold: A parallel simulation framework for multicore systems," in *ISPASS*, March 2014.

[113] Wang, R., Chen, L., and Pinkston, T. M., "Bubble coloring: Avoiding routing- and protocol-induced deadlocks with minimal virtual channel requirement," in *International Conference on Supercomputing*, 2013.

[114] Williams, S., Waterman, A., and Patterson, D., "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[115] Woo, D. H., Seong, N. H., Lewis, D., and Lee, H.-H., "An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth," in *High Performance Computer Architecture (HPCA), 2010*, pp. 1 –12, jan. 2010.

[116] Xie, Y., "Processor architecture design using 3d integration technology," in *VLSI Design, 2010. VLSID '10. 23rd International Conference on*, pp. 446 –451, jan. 2010.

[117] Yang, Z. and Srivastava, A., "Co-placement for pin-fin based micro-fluidically cooled 3d ics," in *ASME 2015 International Technical Conference and Exhibition on Packaging and Integration of Electronic and Photonic Microsystems collocated with the ASME 2015 13th International Conference on Nanochannels, Microchannels, and Minichannels*, pp. V001T09A036–V001T09A036, American Society of Mechanical Engineers, 2015.

[118] Yasuko Eckert, Nuwan Jayasena, G. H. L., "Thermal feasibility of die-stacked processing in memory," December 2014.

[119] Yu, R., "Foundry tsv enablement for 2.5d/3d chip stacking," August 2012.

[120] Yuan, G. L., Bakhoda, A., and Aamodt, T. M., "Complexity effective memory access scheduling for many-core accelerator architectures," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 34–44, ACM, 2009.

[121] Zhang, D., Jayasena, N., Lyashevsky, A., Greathouse, J. L., Xu, L., and Ignatowski, M., "Top-pim: throughput-oriented programmable processing in memory," in *Symposium on high-performance parallel and distributed computing*, 2014.

[122] Zhang, L., Fang, Z., Parker, M., Mathew, B., Schaelicke, L., Carter, J., Hsieh, W., and McKee, S., "The impulse memory controller," *Computers, IEEE Transactions on*, vol. 50, pp. 1117 –1132, Nov. 2001.

[123] Zhang, Y., Dembla, A., Joshi, Y., and Bakir, M. S., "3d stacked microfluidic cooling for high-performance 3d ics," in *Electronic Components and Technology Conference (ECTC), 2012 IEEE 62nd*, pp. 1644–1650, May 2012.

[124] Zhang, Z., Zhu, Z., and Zhang, X., "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," in *33rd annual ACM/IEEE international symposium on Microarchitecture*, pp. 32–41, 2000.

[125] Zheng, H., Lin, J., Zhang, Z., Gorbatov, E., David, H., and Zhu, Z., "Mini-rank: Adaptive dram architecture for improving memory power efficiency," in *41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pp. 210–221, 2008.

[126] Zheng, H., Lin, J., Zhang, Z., and Zhu, Z., "Decoupled dimm: building high-bandwidth memory system using low-speed dram devices," in *36th annual international symposium on Computer architecture*, ISCA '09, pp. 255–266, 2009.

[127] Zheng, L., Zhang, Y., and Bakir, M., "A silicon interposer platform utilizing microfluidic cooling for high-performance computing systems," *Components, Packaging and Manufacturing Technology, IEEE Transactions on*, vol. 5, pp. 1379–1386, Oct 2015.

[128] Zheng, L., Zhang, Y., Huang, G., and Bakir, M. S., "Novel electrical and fluidic microbumps for silicon interposer and 3-d ics," *Components, Packaging and Manufacturing Technology, IEEE Transactions on*, vol. 4, no. 5, pp. 777–785, 2014.