LEVERAGING MIXED PROCESS THREE-DIMENSIONAL DIE STACKING TECHNOLOGY FOR CACHE HIERARCHIES AND RELIABILITY

by

Niti Madan

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

December 2009

Copyright © Niti Madan 2009

All Rights Reserved

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Niti Madan

This dissertation has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

______________ ______________________________
Chair: Rajeev Balasubramonian

______________ ______________________________
John Carter

______________ ______________________________
Ganesh Gopalakrishnan

______________ ______________________________
Erik Brunvand

______________ ______________________________
Srihari Makineni

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the dissertation of Niti Madan in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Rajeev Balasubramonian
Chair: Supervisory Committee

Approved for the Major Department

Martin Berzins
Chair/Dean

Approved for the Graduate Council

Charles A. Wight
Dean of The Graduate School

ABSTRACT

Aggressive technology scaling has led to smaller transistor sizes and steadily improved transistor performance. However, on-chip global wires have not scaled at the same rate causing interconnects to emerge as one of the key bottlenecks. This has an especially pronounced effect on future cache hierarchy design where interconnects exchange data between numerous cores and caches. In addition to rendering processors communication-bound, technology scaling has also decreased processor reliability. Shrinking transistor sizes and lower supply voltages have increased the vulnerability of computer systems towards soft errors (also known as transient faults).

Emerging three-dimensional (3D) integration technology enables vertical stacking of silicon dies, yielding high density and low latency interconnects. This results in increased processor performance as well as reduced power consumption because of shorter on-chip wires. Another salient advantage of 3D stacking is the ability to stack heterogeneous dies that have been fabricated in disparate process technologies. This dissertation explores novel applications for 3D die stacking at the micro-architecture level with an emphasis on this mixed-process integration. We find that reliability and cache-hierarchy design can benefit from 3D die stacking and we explore the various solutions that can help mitigate these challenges in future processor design.

In the first part of this thesis, we propose a reliable processor architecture that uses an in-order checker processor optimized to reduce the power/performance overheads of redundancy. We then leverage the "snap-on" functionality provided by 3D integration and propose implementing this redundant checker processor on a second die. This allows manufacturers to easily create a family of reliable processors without significantly impacting the cost or performance for customers that care

less about reliability. Most importantly, we advocate the use of an older process technology for implementing the checker die that is more error-resilient, showcasing the advantage of mixed-process 3D integration. In the second part of this thesis, we propose the design of a reconfigurable heterogeneous SRAM-DRAM cache. By stacking a DRAM die on top of a SRAM die, the SRAM cache can grow vertically by enabling the DRAM bank above based on the application's working set requirement. This artifact would not have been possible without exploiting mixed-process 3D integration. Such a heterogenous cache design allows us to reap the benefits of DRAM's density and SRAM's superior power and delay characteristics.

In loving memory of my grandfathers:

J.C. Madan and V.P. Joshi

CONTENTS

ABSTRACT . . . . . . **iv**

LIST OF FIGURES . . . . . . **ix**

LIST OF TABLES . . . . . . **xi**

ACKNOWLEDGMENTS . . . . . . **xii**

CHAPTERS

1. INTRODUCTION **1**

1.1 Semiconductor Trends 1
1.2 Leveraging 3D Technology 3
1.3 Dissertation Statement 6
1.4 Dissertation Organization 6

2. BACKGROUND **7**

2.1 Background on 3D Integration Technology 7
2.2 Related Work 8

3. RELIABLE ARCHITECTURES **10**

3.1 Power-efficient Approaches to Reliability 10
3.2 Baseline Reliable Processor Model 13
3.3 Managing Power Overheads 15
3.3.1 Power Reduction Strategies for the Trailing Core 15
3.3.1.1 Dynamic Frequency Scaling (DFS) 16
3.3.1.2 In-Order Execution 17
3.3.1.3 Workload Parallelization 18
3.3.2 Dynamic Frequency Scaling Algorithm for a Single Thread 19
3.3.3 Dynamic Frequency Scaling Algorithm for Multithreaded Workloads 20
3.3.4 Analytical Model 21
3.3.5 Implementation Complexities 25
3.4 Results 26
3.4.1 Methodology 26
3.4.2 Single-Thread Results 29
3.4.3 Multithread Workloads 32
3.4.4 Potential for Voltage Scaling 35

3.4.5 Sensitivity Analysis 38
3.5 Related Work 40
3.6 Summary 43

4. LEVERAGING 3D TECHNOLOGY FOR RELIABILITY 44

4.1 3D Technology for Reliability 45
4.2 Baseline Reliable Processor Model 46
4.3 Proposed 3D Implementation 47
4.3.1 Methodology 48
4.3.2 Thermal Overheads of 3D Checkers 52
4.3.3 Performance 56
4.3.4 Interconnect Evaluation 58
4.3.5 Conservative Timing Margins 60
4.4 The Impact of Heterogeneity on the Checker 62
4.5 Related Work 67
4.6 Summary 67

5. A 3D STACKED RECONFIGURABLE CACHE HIERARCHY DESIGN 69

5.1 Leveraging 3D Technology for Cache Hierarchy Design 70
5.2 Background and Related Work 74
5.3 Proposed Ideas 75
5.3.1 Proposed NUCA Organization 75
5.3.2 Page Coloring 77
5.3.3 Reconfigurable SRAM/DRAM Cache 79
5.3.3.1 Organizing Tags and Data 80
5.3.3.2 Growing Associativity, Sets, Block-Size 81
5.3.3.3 Reconfiguration Policies 83
5.3.4 Interconnect Design 84
5.4 Results 85
5.4.1 Methodology 85
5.4.2 Baseline Organizations 86
5.4.3 Workload Characterization 86
5.4.4 Evaluation of Page Coloring Schemes 87
5.4.5 Reconfigurable SRAM-DRAM Cache 90
5.4.6 Interconnect Evaluation 93
5.5 Summary 94

6. CONCLUSIONS 96

6.1 Future Work 99

REFERENCES 101

LIST OF FIGURES

2.1 An example of F2F bonded 3D stack 8

3.1 An example of the effect of scaling the checker core's frequency. In this example, by operating at half the peak frequency, the checker's dynamic power is reduced from 80W to 40W. Leakage power is not affected. 12

3.2 Baseline chip-level RMT (CRTR) 15

3.3 Power-efficient chip-level RMT design space 16

3.4 IPCs for different models, relative to the baseline out-of-order core. The absolute IPC for the baseline out-of-order core for each program is listed above each set of bars. 29

3.5 Power consumed by the trailing core as a function of the power consumed by the leading core. The number above each set of bars represents the absolute value of power dissipated by the leading core for each benchmark. 31

3.6 Total IPC throughput for multithreaded workloads. 33

3.7 ED^2 for the entire system for various forms of trailers, normalized to the ED^2 of CRTR. 34

3.8 Histogram showing percentage of intervals at each normalized frequency. 37

3.9 Trailer power as a function of contribution of leakage to the baseline processor. 39

3.10 Power overhead of the trailing core, relative to the leading core, with and without parallelizing the verification workload. 39

4.1 Baseline (a) and proposed (c) processor models and example F2F 3D stack (b). Figure not drawn to scale. 49

4.2 Floorplans for each processor model: (a) represents the baseline 2d-a model and the bottom die of the 3d-2a model, (b) represents the upper die of the 3d-2a model, (c) represents the 2d-2a model. 53

4.3 Thermal overhead analysis of 3D checker 54

4.4 Thermal overhead analysis of 3D checker for each benchmark 55

4.5 Performance evaluation (with a NUCA policy that distributes sets across banks). 57

4.6 Histogram showing percentage of intervals at each normalized frequency. 62
4.7 SRAM single-bit soft error scaling rate (data from [1]) 64
5.1 Stacked 3D processor layout. The bottom die contains the 16 cores, the second die contains 16 SRAM banks, and the top die contains 16 DRAM banks. Interdie via pillars (one for each bank) are used to implement vertical connections between cores and DRAM/SRAM banks. Horizontal communication happens on the on-chip tree network (shown on the right) implemented on the second die. There are no horizontal connections between cores (banks) on the bottom (top) die. 72
5.2 Page coloring schemes 78
5.3 Workload characterization: sharing trend in server workloads 87
5.4 Performance evaluation of page coloring schemes as a function of cache capacity 88
5.5 Miss ratio of page coloring schemes as a function of cache capacity . . . 88
5.6 Performance evaluation of SRAM-DRAM cache 90
5.7 Percentage of total time each DRAM bank is switched off for Rp:I+Share4:D. Banks 5, 6, 9, and 10 are the central banks. 91
5.8 Distribution of accesses based on bank number. 92
5.9 Average percentage of hits in SRAM/DRAM ways for R-Rp:I+Share4:D. 93
5.10 Average percentage of hits in SRAM/DRAM ways for NR-Rp:I+Share4:D. 93
5.11 Distribution of bank accesses based on distance 94

LIST OF TABLES

3.1 Fetch and frequency policies adopted for leading and trailing cores in P-CRTR. For each entry in the grid, the first two terms indicate which thread is fetched for core 1 and core 2, and the last term indicates the frequency selected for the trailing core. Fetch policies are adjusted every cycle and frequencies can be adjusted at intervals of 1000 cycles. 22

3.2 Simplescalar simulation parameters 27

3.3 Benchmark pairs for the multithreaded workloads. 28

4.1 Thermal model parameters 50

4.2 Area and power values for various blocks 51

4.3 Simplescalar simulation parameters 51

4.4 D2D interconnect bandwidth requirements 59

4.5 Impact of pipeline scaling on power overheads in terms of baseline dynamic power consumption 61

4.6 Impact of technology scaling on variability 64

4.7 Impact of technology scaling on various device characteristics 65

4.8 Impact of technology scaling on power consumption 65

5.1 Access time, energy, and area for various cache configurations at a 4 GHz clock derived from [2]. 82

5.2 Simulation parameters 85

ACKNOWLEDGMENTS

I would like to first sincerely thank Rajeev for being an incredible advisor. He taught me how to conduct quality research. He not only set the bar high but also helped me get there. Rajeev's patience and encouragement helped me cope with difficult times when my initial research efforts were not working out that well. Apart from guiding me in research, he always gave valuable feedback on how to succeed in a research career. He also gave me the flexibility of working remotely out of California. Rajeev, I cannot thank you enough for this flexibility, which helped me balance my personal life and work. I would also like to thank my thesis committee members Ganesh Gopalakrishnan, Erik Brunvand, John Carter and Srihari Makineni for their constructive and timely feedback on my research. I am also grateful to Li Zhao, Ravi Iyer and Srihari for their help and feedback especially on the DRAM cache project during my internship at Intel Research. A very special thanks to my graduate school coordinator - Karen Feinauer - for always helping out with so much paperwork in a timely way.

Many thanks to my Utah colleagues - Naveen, Karthik, Manu, Vivek, Dave, Kshitij, Seth, Aniruddha, Byong and Devyani - for their friendship and support during my graduate school years. I have bugged some of you a lot for helping me with simulators. I am also grateful to many friends of mine from Utah for making this experience so much more memorable. Thanks to Chitra, Piyush, Amit, Ruchika, Mohit and Nidhi for making me feel at home in Utah. I will never forget the way each one of you has pampered me whenever I was visiting Utah for work. Special thanks to my friends Geeta, Abhijeet, Sonia, Vidya and Satarupa for your encouragement and support and listening to my graduate school gripe.

I would like to thank my parents for being my role models in pursuing a doctorate in science and engineering. Without your inspiration and encouragement,

I would have never taken on this path. Thanks to my brother Vikram and my in-laws for their support. I am also very grateful to my loving grandparents for their love and blessings. Thanks to my dear son Roshan who has motivated me to write this dissertation even before he was born. Finally, I would like to thank my husband Shashi for always being there for me and motivating me constantly. Thanks for all the sacrifices that you made for me and for never complaining whenever I would ruin your weekend plan because of my deadline pressures.

CHAPTER 1

INTRODUCTION

1.1 Semiconductor Trends

Process technology, circuit, and architectural innovations have continued to steadily improve processor performance over the past many years. The shrinking of transistor sizes has increased the transistor density, resulting in increasing chip functionality and faster circuits. However, this aggressive scaling has introduced a new set of challenges that need to be addressed. Some of these challenges faced by the semiconductor industry are:

- Wire delay: While logic gates have become faster due to smaller transistor sizes, the same does not hold true for on-chip interconnects. The resistance per unit length of wire increases quadratically as a function of reduction in transistor feature width. Also, the capacitance per unit length of wire increases due to increase of coupling capacitance between neighboring wires. This causes the overall delay for a fixed-length wire to increase as the feature width decreases. This is a major problem for cross-chip global wires, especially if combined with increases in clock frequency. To alleviate the quadratic dependence of wire delay on wire length, these wires are broken down into smaller segments by inserting repeaters between them. This allows a linear dependence of delay on wire length. However, these repeaters end up increasing the power consumption of wires. Therefore, on-chip wires have become a big bottleneck from performance and power viewpoint in modern microprocessors. With the advent of multicore era and large caches required to feed many cores, interconnects comprise a significant portion of a chip's real-estate. Thus, modern processors have become increasingly communication bound [3, 4].

- Reliability: Shrinking transistor sizes and lower supply voltages have led to increased transistor reliability failures [5]. Firstly, radiation and alpha particles in semiconductor material can cause enough charge to be deposited in a transistor causing a bit-flip. This may result in logical errors in program computation but does not cause any permanent device damage. Such errors are refered to as soft errors or transient errors. It has been projected that these transient errors are expected to increase exponentially in modern microprocessors [6]. Secondly, parameter variation can cause uncertainties in various device dimensions (such as gate length/width, wire spacing/height/width), that can influence the delay and power consumption of circuits. Dynamic conditions such as temperature, supply voltage noise, and cross-coupling effects can also influence circuit delay. Because of these variations, timing constraints are occasionally not met leading to latching of incorrect values at the end of that clock cycle. These error rates may increase as the processor ages and different devices wear out at different rates because of phenomena such as NBTI [7].We refer to these errors as dynamic timing errors. Thirdly, smaller transistors and high on-chip power densities have accelerated processor wearout causing permanent device failures known as hard errors [7]. Thus, reliability has become a first-class design constraint in modern processor design.

- Power consumption: Dynamic power consumed by a chip is linearly dependent upon operating clock frequency, transistor capacitance and quadratically upon supply voltage. Even though technology scaling has enabled lower supply voltage, the actual supply voltage utilized is much higher due to increased reliability failures as described above. Also, interconnects in modern processors may consume 50% of total chip power [8]. Apart from dynamic power consumption, leakage power poses another serious challenge to designers. In order to reap the performance benefit of technology scaling, threshold voltages need to be scaled down proportional to supply voltage. However, leakage current is inversely dependent upon threshold voltage. Also, thinning

of gate-oxides have led to increased gate leakage currents. Thus, leakage power is catching up with dynamic power consumption. Several solutions at manufacturing process, circuit and architecture level have been proposed to push the power budget in modern microprocessors [9–11].

1.2 Leveraging 3D Technology

At the architecture level, several solutions are being studied to deal with all the above challenges. The goal of this thesis is to further the state-of-the-art and research and explore opportunities to handle these challenges by leveraging emerging technologies. We especially focus on emerging 3D technology and exploit its advantages from a computer architect's perspective to elegantly solve some of these problems.

Emerging 3D die stacking [12] appears promising and has already been employed in the embedded domain [13, 14]. 3D integration enables vertical stacking of multiple dies that communicate with each other using low-latency and high-bandwidth vertical interconnects (also known as die to die vias, interdie vias or interdie pillars). Three primary advantages of 3D technology from a computer architect's perspective are as follows:

- Stacking of heterogeneous dies: A concrete application of this approach is the 3D stacking of DRAM chips upon large-scale CMPs. Interdie vias can take advantage of the entire die surface area to implement a high bandwidth link to DRAM, thereby addressing a key bottleneck in CMPs that incorporate nearly a hundred cores [13, 15].

- "Snap-on" analysis engines: Chips employed by application developers can be fitted with additional stacked dies that contain units to monitor hardware activity and aid in debugging [16]. Chips employed by application users will not incorporate such functionality, thereby lowering the cost for these systems.

- Improvement in CPU performance/power: The components of a CPU (cores and cache banks, pipeline stages, individual circuits) can be implemented

across multiple dies. By lowering the penalties imposed by long wires, performance and power improvements are possible.

However, one salient disadvantage of 3D technology is the effect on a chip's thermals. Dies that are stacked further away from the heatsink are unable to dissipate heat efficiently leading to high thermal overheads. Given that power and thermals already pose a significant challenge to modern processors, it is imperative that 3D chips not push the power/thermal envelope further. Therefore, to take advantage of 3D technology, architects need to ensure that stacked dies are not power-hungry.

We leverage some of the above-mentioned advantages and propose novel applications of 3D technology that alleviate reliability challenges and on-chip communication issues especially in cache hierarchies. Our proposals ensure that the thermal overheads of 3D stacking are within the tolerable range.

We first propose a power-efficient reliable architecture that is later utilized for building a 3D reliable processor. Since most reliability solutions employ some form of redundancy, we find that most of these solutions incur significant power and area overheads. We focus on one such technique - redundant multithreading (RMT) that employs a redundant thread or core to recompute an instruction's output and verify all instructions before committing the final output. We optimize RMT technique for balancing power, performance, and area overheads of redundancy and propose in-order execution of the redundant core (also known as checker core) combined with aggressive value speculation. We find that such a checker core design is an elegant fit for 3D stacking due to its power-efficiency.

We next demonstrate the application of 3D stacking for improving processor reliability. We propose stacking the power-efficient redundant core on top of the main core. All of the above-mentioned advantages of 3D stacking can be exploited if we stack the checker core on a separate die. (i) The communication of results between the two cores is efficient in terms of delay and power as shorter die-to-die interconnects can be utilized. By isolating the checker core on a separate die, layout and wiring of the main core will require minimal modifications. (ii) Chip

manufacturers can individually manufacture main core die and checker die and can offer products with varying number of checker dies depending upon the reliability requirement. This exploits the snap-on functionality of 3D technology where users requiring minimal reliability support need not bear the cost of an additional checker die. (iii) Finally, reliability makes a compelling application for stacking heterogeneous dies: the checker die can be fabricated in an older error-resilient process. We also show that an older process helps reduce the thermal overheads due to lower power-density enabled by larger transistor sizing.

Finally, we explore the design of a 3D reconfigurable heterogeneous cache hierarchy that optimizes communication and capacity in large last level caches. We propose stacking DRAM over SRAM to design a heterogeneous SRAM-DRAM cache. This allows the SRAM bank to grow vertically and enable the DRAM bank directly above to form a large SRAM-DRAM cache slice for applications that require large working set size. With this implementation, we can take advantage of higher density of DRAM (8x compared to SRAM) and at the same time benefit from superior delay/power characteristics of SRAM. Thus, leveraging mixed process 3D integration by stacking DRAM die over SRAM die, enables us to design a reconfigurable heterogeneous cache, an artifact that would not have been possible in a traditional 2D implementation. While this approach helps optimize capacity in large caches, we also explore other techniques such as OS-based page-coloring and on-chip tree network design that help optimize horizontal communication between cache banks in multithreaded applications. We find that the combination of these techniques leads to a balanced cache hierarchy design that can scale across many cores.

This thesis encompasses various proposals that emphasize the importance of mixed-process 3D integration. We show that the stacking of a checker core in an older error-resilient process and design of a SRAM-DRAM hybrid cache are novel and concrete applications of this technology.

1.3 Dissertation Statement

Aggressive technology scaling has led to increased processor reliability failures and exacerbating power and delay issues in on-chip interconnects. We believe that emerging 3D technology can help alleviate some of these challenges. In this thesis, we demonstrate how mixed-process 3D integration can be leveraged to design future reliable processors and efficient cache hierarchies.

1.4 Dissertation Organization

This dissertation has been organized as follows. Chapter 2 gives a brief background on 3D integration and discusses prior art in 3D architectures. Additional related work is described in later chapters after describing our innovations. Chapter 3 explores techniques to reduce power and performance overheads of reliable architectures especially redundant multithreading alternatives. In Chapter 4, we leverage the proposed power-efficient reliable architecture proposed in Chapter 3 and propose a 3D stacked reliable processor. Chapter 5 proposes the design of a 3D reconfigurable cache hierarchy and we finally draw conclusions in Chapter 6.

CHAPTER 2

BACKGROUND

2.1 Background on 3D Integration Technology

In this section, we present an overview of 3D integration technology. There are various types of vertical interconnects that can be utilized for 3D integration such as wire-bonded, contactless, micro-bump and through-silicon-via (TSV) [17]. Amongst all these alternatives, TSV is the most promising with respect to scaling of interconnect density. As a result, many recent studies have focused on TSV technology for 3D stacking and the following two implementation approaches have been explored:

- Bottom-up Approach: This approach involves sequential device fabrication process such as Multilayer Buried Structures (MLBS). In this implementation, each active device layer is first sequentially fabricated on the wafer using the frontend processing and then interconnects are added to all these multiple layers using the backend processing.

- Top-down Approach: This approach involves separately fabricating each device layer or die. Each of these device layers/dies are then assembled to build the 3D chip by using wafer bonding technology. Wafers can be bonded either as face-to-face (F2F) or face-to-back (F2B) or back-to-back (B2B). Note that TSVs do not go through thick silicon layers if F2F bonding (illustrated in Figure 2.1) is employed and thus can use smaller via sizes. However, F2F bonding doesn't scale beyond two dies and for aggressively stacked 3D chips with multiple dies, B2B bonding is inevitable.

Between the above two approaches, top-down wafer bonding approach requires minimal changes to the current manufacturing process whereas MLBS will require

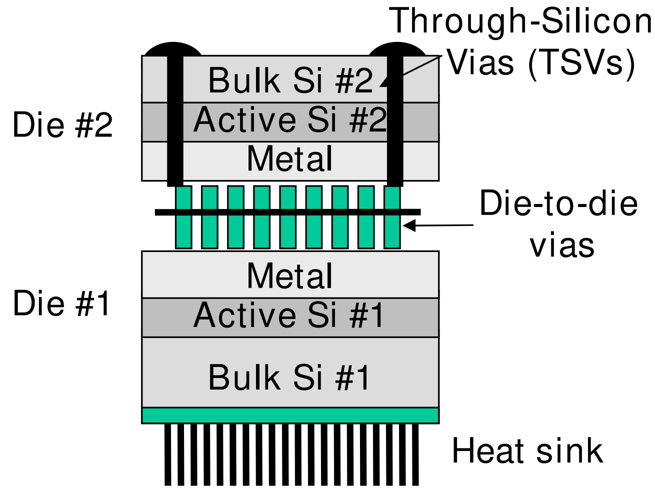


Figure 2.1. An example of F2F bonded 3D stack

expensive process upgrades. However, in terms of via size scaling, interconnect in MLBS is able to scale better as it uses the same process feature size for local wires. Wafer bonding can have via scaling restrictions due to wafer alignment tolerance issues. Therefore, more fine-grained partitioning of architectural resources is easier in MLBS process. Since manufacturing cost is a bigger constraint in adoption of 3D technology, many studies focus on wafer bonding techniques for implementing our 3D chips.

The state-of-the-art 3D technology allows die-to-die (D2D) vias or TSVs to be very small with 4-10μm [18] pitch yielding an overall density of up to 1000,000 vias per cm^2. These vias are not only small but also fast. It has been reported that the time taken to traverse 20 dies is only 12 ps [19].

2.2 Related Work

There has been much recent work on architectural evaluations involving 3D chips. Many have focused on improving single-core performance and power [12, 20–22] by stacking a traditional processor pipeline's 2D structures across multiple dies. A few studies have also explored splitting/folding individual pipeline structures such as SRAM arrays in register files [23] and adder cells in functional units [24]

between multiple dies to reduce the wire delay in these critical structures. These solutions stack 3D implementations of processor structures.

Some studies [22, 25]have focused on reducing the thermal overheads of 3D stacking while a few [26, 27] have investigated 3D network-on-chip architectures.

Mysore et al. [16] were one of the first to exploit the snap-on advantage of 3D technology at the architecture level. They proposed stacking an analysis and software profiling engine on top of a regular cpu to aid software debug professionals. In our work [28], we propose a reliable 3D processor that stacks a checker core to provide a snap-on reliability engine. Many of the findings of our study are specific to the implementation of a checker core on the second die: most notably, the effect of technology, pipelining, and frequency scaling on the ability of the checker core to resist transient and dynamic timing errors. Apart from differences in the applications studied for snap-on functionality, Mysore et al. [16] do not explore microarchitectural choices for their analysis engine. Zhang et al. [29] proposed a follow-on work to our 3D reliability solution that takes advantage of error resilience and shielding effect of different layers in a 3D stack. Our body of work was the first one to propose the use of a heterogeneous older process for reliability whereas [29] takes this one step further and studies the impact of error-tolerant Silicon-On-Insulator (SOI) process for enhanced reliability.

A number of recent papers have proposed cache hierarchy organizations in 3D. Li et al. [30] describe the network structures required for the efficient layout of a collection of cores and NUCA cache banks in 3D. Some bodies of work implement entire SRAM or DRAM cache structures on separate dies [19, 20, 31, 32]. Loh [33] proposes the changes required to the memory controller and DRAM architecture if several DRAM dies are stacked upon the CPU die to implement main memory. Ours is the first body of work that proposes reconfiguration across dies and combines heterogeneous technologies within a single level of cache.

CHAPTER 3

RELIABLE ARCHITECTURES

In one of the key proposals of this thesis, we advocate the use of 3D die stacking for implementing a reliable processor by stacking redundant hardware on the second die. However, in most implementations, power overheads of redundancy are typically very high. In order to design a 3D reliable processor, we first explore novel techniques to improve power-efficiency of redundant cores such that thermal overheads of 3D stacking are tolerable. This chapter highlights solutions towards this direction.

This chapter is organized as follows. We first make a case for power-efficiency to be an important design constraint for reliability techniques in Section 3.1. Section 3.2 describes the redundant multithreading implementations that serve as baseline processor models in this study. Section 3.3 describes various power reduction strategies for the trailing thread. The proposed ideas are evaluated in Section 3.4. Section 3.5 outlines the relationship of this work with prior art and we summarize the conclusions of this study in Section 3.6.

3.1 Power-efficient Approaches to Reliability

A recent study [6] shows that the soft-error rate per chip is projected to increase by nine orders of magnitude from 1992 to 2011. This is attributed to growing transistor densities and lower supply voltages that increase susceptibility to radiation and noise. Such soft errors or transient faults do not permanently damage the device, but can temporarily alter state, leading to the generation of incorrect program outputs.

Fault tolerance can be provided at the circuit or process level. For comprehensive fault coverage, every circuit would have to be re-designed. This not only

increases design complexity, but also has the potential to lengthen critical paths and reduce clock frequencies. For this reason, many recent studies [34–40] have explored architecture-level solutions that can provide fault tolerance with modest performance and complexity overheads. In most solutions, generally referred to as redundant multithreading (RMT), an instruction is executed twice and results are compared to detect faults. Most studies on reliability have paid little attention to power overheads in spite of the fact that future microprocessors will have to balance three major metrics: performance, power, and reliability. A recent paper by Gomaa and Vijaykumar [41] opportunistically employs redundancy, thereby deriving a desireable point on the performance-reliability curve. Because redundancy is occasionally turned off, this approach also indirectly reduces power overheads. In this study, we focus on maintaining a constant level of error coverage and explore different strategies to improve the power-efficiency of reliability mechanisms (while occasionally compromising marginal amounts of performance).

In a processor that employs redundancy, the *"checker instruction"* can be made to flow through a similar pipeline as the *"primary instruction"*. [1] This approach is well suited to a chip multiprocessor (CMP) or simultaneous multithreaded processor (SMT), where the processor is already designed to accommodate multiple threads. With minor design modifications, one of the thread contexts can be made to execute the checker thread [35, 36, 38, 40]. Further, thread contexts can be dynamically employed for either checker or primary threads, allowing the operating system or application designer to choose between increased reliability or increased multithreaded performance. However, this approach has significant power overheads as each checker instruction now flows through a complex out-of-order pipeline. In an alternative approach, the checker thread can flow through a heavily modified helper pipeline that has low complexity [34, 39]. Even though the area overhead is modest, the area occupied by this helper pipeline is not available for use by primary

[1] The main program thread is referred to as the *primary* or *leading thread*. The redundant thread is referred to as the *checker* or *trailing thread*. Correspondingly, these threads execute on *primary/leading cores* or *checker/trailing cores*.

threads even if reliability is not a primary concern for the application. As we shall show in this chapter, heterogeneous CMPs can allow us to derive the best of the two approaches above.

As a starting point, we consider the following RMT architecture based on the Chip-level Redundantly Threaded multiprocessor with Recovery (CRTR) model proposed by Gomaa et al. [35]. The primary thread executes on an out-of-order core and the checker thread executes on a different out-of-order core within a CMP. Branch outcomes, load values, and register results produced by the primary thread are fed to its checker thread in the neighboring core so it can detect and recover from faults (as shown in Figure 3.1). In an effort to reduce the power overhead of CRTR, we make the following observations. The checker thread experiences no branch mispredictions or cache misses because of the values fed to it by its primary thread. The checker thread is therefore capable of a much higher instruction per cycle (IPC) throughput rate than its primary thread. This allows us to operate the checker core in a low-power mode, while still matching the leading thread's throughput. Figure 3.1 shows how the checker core's frequency can be scaled down in order to reduce dynamic power. We also explore the potential of using an in-order pipeline for the checker core and show that some form of value prediction is required to enable it to match the throughput of the primary thread. We also

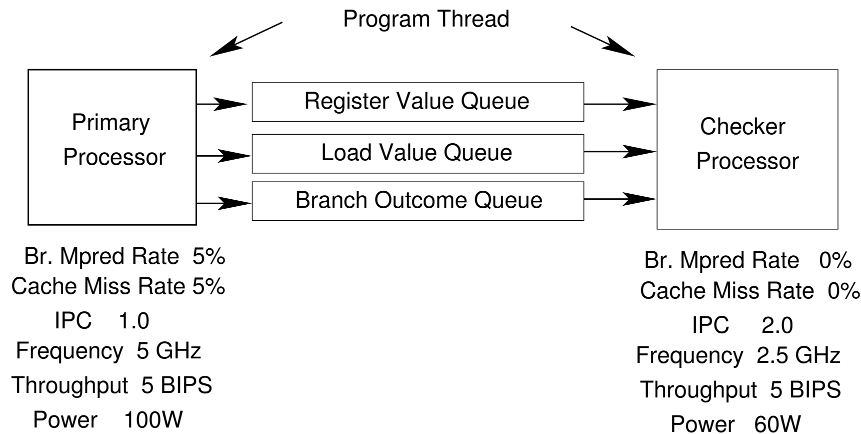


Figure 3.1. An example of the effect of scaling the checker core's frequency. In this example, by operating at half the peak frequency, the checker's dynamic power is reduced from 80W to 40W. Leakage power is not affected.

extend our evaluation to multithreaded workloads executing on a CMP of SMTs. Finally, we examine the potential of dynamic voltage scaling and of parallelization of the verification workload. Some of the conclusions of this work resonate well with prior research, such as the proposal by Austin to employ in-order checker pipelines that are fed with leader-generated inputs [34]. On the other hand, some of our conclusions argue against the voltage scaling approach proposed by Rashid et al. [37].

3.2 Baseline Reliable Processor Model

We have based our reliable chip-multiprocessor architecture on the model proposed by Gomaa et al. [35] and Mukherjee et al. [36]. The architecture consists of two communicating cores that execute copies of the same application for fault-detection. One of the cores (the leading core) executes ahead of the second core (the trailing core) by a certain amount of slack. The leading core communicates its committed register results to the trailing core for comparison of values to detect faults (Figure 3.1). Load values are also passed to the trailing core so it can avoid reading values from memory that may have been recently updated by other devices. Thus, the trailing thread never accesses its L1 data cache and there is no need for coherence operations between the L1 data caches of the leading and trailing cores. This implementation uses *asymmetric commit* to hide intercore communication latency – the leading core is allowed to commit instructions before checking. The leading core commits stores to a store buffer (StB) instead of to memory. The trailing core commits instructions only after checking for errors. This ensures that the trailing core's state can be used for a recovery operation if an error occurs. The trailing core communicates its store values to the leading core's StB and the StB commits stores to memory after checking.

The communication of data between the cores is facilitated by the first-in-first-out Register Value Queue (RVQ) and Load Value Queue (LVQ). As a performance optimization, the leading core also communicates its branch outcomes to the trailing core (through a branch outcome queue, BOQ), allowing it to have perfect branch

prediction. The power saved in the trailing core by not accessing the L1D cache and branch predictor is somewhat offset by the power consumption of the RVQ and LVQ. If the slack between the two cores is at least as large as the re-order buffer (ROB) size of the trailing core, it is guaranteed that a load instruction in the trailing core will always find its load value in the LVQ. When external interrupts or exceptions are raised, the leading thread must wait for the trailing thread to catch up before servicing the interrupt.

The assumed fault model is exactly the same as in [35, 36]. The following conditions are required in order to detect and recover from a single fault:

- The data cache, LVQ, and buses that carry load values must be ECC-protected as the trailing thread directly uses these load values.

- When an error is detected, the register file state of the trailing thread is used to initiate recovery. The trailing thread's register file must be ECC-protected to ensure that values do not get corrupted once they have been checked and written into the trailer's register file.

Other structures in each core (including the RVQ) need not have ECC or other forms of protection as disagreements will be detected during the checking process. The BOQ need not be protected as long as its values are only treated as branch prediction hints and confirmed by the trailing pipeline. Similar to the baseline model in [35, 36], we assume that the trailer's register file is not ECC-protected. Hence, a single fault in the trailer's register file can only be detected. [2] All other faults can be detected and recovered from. The proposed mechanisms in this paper preserve this basic fault coverage.

In our single-thread model, we assume an implementation where each core on the CMP can support only a single thread. Our multithread model is based on the CRTR architecture [35], where each core is a dual-threaded SMT. In the CRTR architecture, the trailing thread of one application shares its core with the leading

[2] If no ECC is provided within the register file, Triple Modular Redundancy will be required to detect and recover from a single fault.

thread of a different application (shown in Figure 3.2). We require that the slack for each application remain between two thresholds, $TH1$ and $TH2$. The lower threshold $TH1$ is set to the ROB size available to the trailing thread so that load results can be found in the LVQ. The higher threshold $TH2$ is set to the size of the RVQ minus the ROB size for the leading thread so that all completing instructions in the leading thread are guaranteed to find an empty slot when writing results into the RVQ. Similar to the fetch policy in [35], the slack values determine which threads are allowed to fetch within each SMT core. If the slack for an application is less than $TH1$, the trailing thread is not allowed to fetch and if the slack is greater than $TH2$, the leading thread is not allowed to fetch. In cases where both threads within an SMT core are allowed to fetch, the ICOUNT heuristic [42] is employed.

3.3 Managing Power Overheads

3.3.1 Power Reduction Strategies for the Trailing Core

For a single-thread workload, we propose that the leading and trailing thread execute on neighboring cores. If each core has SMT capability, it is possible to execute the leading and trailing threads on a single core – this avoids the overhead of intercore communication. However, as we will show later, the power savings possible by executing the trailer on a neighboring core are likely to offset the power overheads of intercore communication.

For a multithreaded workload, the CRTR implementation executes unrelated leading and trailing threads on a single SMT core (Figure 3.2). Since the trailing thread never executes wrong-path instructions and never accesses the data

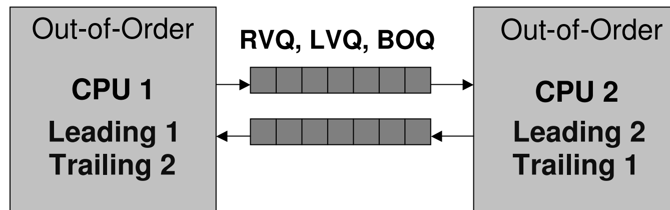


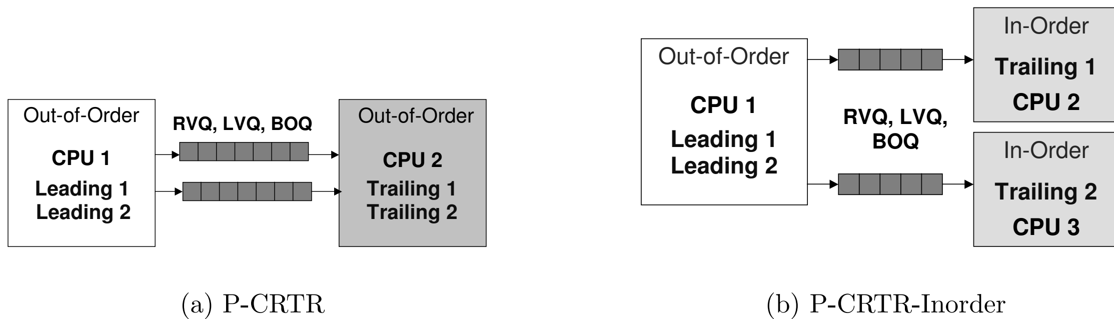
Figure 3.2. Baseline chip-level RMT (CRTR)

cache, the leading thread that executes in tandem is likely to experience little contention, thereby yielding high throughputs. Applying a power-saving strategy to a trailing thread in this setting will slow the leading thread that executes on that same core. Hence, to enable power optimizations, we propose executing two leading threads on the same SMT core and the corresponding trailing threads on neighboring cores, referred to as Power-efficient CRTR (P-CRTR) (Figures 3.3a and 3.3b). By changing the assignment of threads to cores, we are not increasing intercore bandwidth requirements – the same amount of data as in CRTR is being communicated between cores. Since the leading threads compete for resources within a single core in P-CRTR, a performance penalty is incurred.

An RMT system attempts to maintain a roughly constant slack between leading and trailing threads. Since the trailing thread benefits from perfect caching and branch prediction, it tends to catch up with the leading thread. This provides the opportunity to throttle the execution of the trailer in a manner that lowers its power consumption and maintains the roughly constant slack.

3.3.1.1 Dynamic Frequency Scaling (DFS)

The first approach we consider to throttle trailing thread execution is Dynamic Frequency Scaling (DFS), a well-established technique that allows dynamic power to scale down linearly with clock frequency [43]. It is a low overhead technique – in Intel's Montecito, a frequency change can be effected in a single cycle [43]. In most



(a) P-CRTR

(b) P-CRTR-Inorder

Figure 3.3. Power-efficient chip-level RMT design space

systems, DFS does not result in a dynamic energy reduction as execution time is also linearly increased. As we show in our results, the application of DFS to the trailing core of an RMT system has a minimal impact on execution time. Hence, in this particular case, DFS results in a reduction in dynamic power *and* dynamic energy. DFS does not impact leakage power (and leakage energy) dissipated by the trailer.

A competitive alternative to DFS is run-and-stall, where the trailing thread operates at peak frequency for a while and then shuts off its clock for a while. We expect that the fraction of stall time in run-and-stall will equal the average frequency reduction in a DFS-based mechanism. Run-and-stall will also not impact leakage unless voltage is turned off during the stall (voltage changes are known to have much higher delay overheads). Given the similarity with DFS, we do not further evaluate run-and-stall in this paper.

3.3.1.2 In-Order Execution

When throttling the trailing core, we may see greater power savings by executing the trailing thread on an in-order core. A short and simple pipeline can have a significant impact on both dynamic and leakage power. Unfortunately, for many program phases, an in-order core, even with a perfect D-cache and branch predictor, cannot match the throughput of the leading out-of-order core. Hence, some enhancements need to be made to the in-order core. We considered the effect of increasing fetch bandwidth and functional units in a simple in-order core but that did not help match the leading core's throughput. In fact, for our simulation parameters, even with a perfect cache and branch predictor, doubling the fetch bandwidth and the ALUs of the in-order core resulted in only about a 1% performance improvement. This is not surprising because data dependency is the biggest bottleneck that limits ILP in an in-order core – as soon as a pair of dependent instructions is encountered, the dependent instruction (and every instruction after it) is forced to stall at least until the next cycle, regardless of the fetch and ALU bandwidth.

We therefore propose the use of register value prediction (RVP). Along with the result of an instruction, the leading thread can also pass the input operands for that instruction to the trailing thread. Instructions in the trailing core can now read their input operands from the RVQ instead of from the register file. Such a register value predictor has a 100% accuracy in the absence of soft errors. With perfect RVP, instructions in the trailer are never stalled for data dependences and ILP is constrained only by a lack of functional units or instruction fetch bandwidth. The value predictions must be confirmed by the trailing core, else an error in the leading core may go un-detected. When an instruction in the trailing core is committed, the trailing register file is read to confirm that the input operands match those in the RVQ. The error coverage is exactly as before – a single soft error in the trailer's register file can be detected, while a single soft error elsewhere can be detected and recovered from. While RVP entails some changes to the commit pipeline in the trailer, it allows us to leverage the benefits of an in-order core.

It may be possible to even apply DFS to the in-order core to further reduce dynamic power. Contrary to previous studies [35] that attempt to reduce intercore traffic, we believe it may be worthwhile to send additional information between cores because of the power optimizations it enables at the trailer.

The use of an in-order core has another major advantage (not quantified in this paper). Since an in-order core has less speculative state, it requires fewer rename registers. It may be possible to include ECC within the in-order core's small register file and still meet cycle time constraints. As discussed in Section 3.2, a trailing core with an ECC-protected register file has higher error recovery coverage.

3.3.1.3 Workload Parallelization

Assuming that power is a quadratic function of performance [44] and that a workload can be perfectly parallelized, it is more power-efficient to execute the workload in parallel across N low-performance cores than on a single high-performance core. The trailing thread is an example of such a highly parallel workload [37]. At regular intervals, the leading thread spawns a new trailing thread that verifies the

results for a recently executed contiguous chunk of the program. At the start of the interval, initial register state must be copied into the trailing core. In the next section, we show that such an approach yields little benefit.

3.3.2 Dynamic Frequency Scaling Algorithm for a Single Thread

The power-efficient trailing cores rely on a DFS mechanism to match their throughputs to that of the leading core. For a single-thread workload, the goal of the DFS mechanism is to select a frequency for the trailing thread so that a constant slack is maintained between the leading and trailing thread. In essence, if the IPC of the leading thread is denoted by IPC_L, and the IPC of the trailing thread is IPC_T, we can maintain equal throughputs and constant slack by setting the trailing core's frequency f_T to $f_L \times IPC_L/IPC_T$, where f_L is the leading core's frequency. The same effect can be achieved with a simple heuristic that examines the size of the buffer that feeds results from the leading to the trailing thread (for example, the RVQ).

Initially, the trailing core is stalled until the leading core commits N instructions. At this point, an RVQ (that does not filter out register values) will have N entries. The trailing core then starts executing. The RVQ is checked after a period of every T cycles to determine the throughput difference between the two cores. If the RVQ has $N - thresh$ entries, it means the trailing thread is starting to catch up with the leading thread. At this point, the frequency of the trailing thread is lowered, one step at a time. If the RVQ has $N + thresh$ entries, it means the leading thread is starting to pull away and the frequency of the trailing thread must be increased. We observed that increasing the frequency in steps causes the slack to increase drastically as the leading thread continues to extend its lead over subsequent intervals. Note that the leading thread has just entered a high IPC phase of the program, while the trailing thread will have to commit N more instructions before it enters that phase itself. Once all the queues are full, the leading thread is forced to stall. To minimize this occurrence, the frequency of the trailing thread

is immediately increased to the leading thread's peak frequency if the RVQ has $N + thresh$ entries.

The smaller the value of T, the faster the mechanism reacts to throughput variations. For our simulations, we conservatively assume a 10 cycle overhead for every dynamic frequency change. The time interval T is selected to be 1000 cycles so that the overhead of frequency scaling is marginal. To absorb throughput variations in the middle of an interval, a slack of 1000 is required. To ensure that the RVQ is half-full on average, we set $N - thresh$ to be 400 and $N + thresh$ to be 700. Frequency is reduced in steps that equal $f_L \times 0.1$.

3.3.3 Dynamic Frequency Scaling Algorithm for Multithreaded Workloads

If each trailer executes on a separate core (as in Figure 3.3c), the single-thread DFS algorithm from the previous subsection can be applied to tune the frequency of each trailing core. If two trailing threads execute on a single SMT core (as in Figure 3.3b), the DFS algorithm will have to consider the slack for both threads in determining the core frequency. We employ fetch throttling strategies to accommodate the potentially conflicting needs of co-scheduled threads. Rather than always use ICOUNT as the fetch policy for the trailing core, it helps to periodically throttle fetch for the thread that has a lower IPC_L/IPC_T ratio and give a higher priority to the other trailing thread. This further boosts the IPC value for the other trailing thread, allowing additional frequency reductions.

The detailed algorithm for invoking fetch throttling and frequency scaling is formalized in Table 3.1. To allow fine-grained control of each thread, we employ three slack thresholds. The action for each case is based on the following guidelines: (i) if slack for a trailer is less than $TH0$, there is no point fetching instructions for that trailer, (ii) if slack for a trailer is between $TH0$ and $TH1$ (the desireable range), the decision depends on the state of the other thread, (iii) if slack is between $TH1$ and $TH2$, we may need to quickly ramp up the frequency to its peak value in an effort to keep slack under control, (iv) if slack is greater than

$TH2$, we can stop fetching instructions for the leader. Table 3.1 describes the action taken for every combination of slack values for both threads. As before, $TH0$ is a function of the trailing thread's ROB size, $TH2$ is a function of the sizes of the RVQ and leader's ROB size, and $TH1$ is picked so that the RVQ will be half-full, on average. The leading core always employs the ICOUNT fetch heuristic to select among the two leading threads unless one of the slacks is greater than $TH2$. Fetch throttling is a low-overhead process and can be invoked on a per-cycle basis. Slack values are evaluated every cycle and any changes to the fetch policy are instantly implemented. Changes in frequency are attempted only every 1000 cycles to limit overheads. The above algorithm has been designed to react quickly to changes, so as to minimize stalls for leading threads. This leads to frequency changes in almost every interval, unless the peak or lowest frequency is being employed. Incorporating some hysteresis in the algorithm reduces the frequency change overhead, but introduces additional stalls for leading threads.

3.3.4 Analytical Model

To better articulate the factors that play a role in overall power consumption, we derive simple analytical power models for the proposed RMT systems. These models also allow an interested reader to tweak parameters (contribution of leakage, ratio of in-order to out-of-order power, etc.) and generate rough estimates of power overheads without detailed simulations. The analytical equations were derived after studying the detailed simulation results described in the next section. We found that when various parameters were changed, our detailed simulation results were within 4% of the analytical estimates.

For starters, consider leading and trailing threads executing on neighboring out-of-order cores in a CMP. Assuming that the leader has $wrongpath_factor$ times the activity in the trailer (because of executing instructions along the wrong path), the total power in the baseline RMT system is given by the following equation.

$$\begin{aligned} Baseline_power = leakage_{leading} + dynamic_{leading} + \\ leakage_{leading} + dynamic_{leading}/wrongpath_factor \end{aligned} \tag{3.1}$$

Table 3.1. Fetch and frequency policies adopted for leading and trailing cores in P-CRTR. For each entry in the grid, the first two terms indicate which thread is fetched for core 1 and core 2, and the last term indicates the frequency selected for the trailing core. Fetch policies are adjusted every cycle and frequencies can be adjusted at intervals of 1000 cycles.

l_1 = leading thread for application 1; l_2 = leading thread for application 2
t_1 = trailing thread for application 1; t_2 = trailing thread for application 2
s_1 = slack for application 1; s_2 = slack for application 2
l_1 and l_2 execute on core 1; t_1 and t_2 execute on core 2
The fetch and frequency policy attempt to maintain a slack between $TH0$ and $TH1$.
In our simulations, $TH0$ is set to 80, $TH1$ is set to 700, $TH2$ is set to 1000.

Slack conditions	$s_2 \leq TH0$	$TH0 < s_2 \leq TH1$	$TH1 < s_2 \leq TH2$	$TH2 < s_2$
$s_1 \leq TH0$	ICOUNT : stall f_t $-=$ $0.1f_{peak}$	ICOUNT : t_2 f_t $-=$ $0.1f_{peak}$	ICOUNT : t_2 f_t $+=$ $0.1f_{peak}$	$l_1 : t_2$ $f_t = f_{peak}$
$TH0 < s_1 \leq TH1$	ICOUNT : t_1 f_t $-=$ $0.1f_{peak}$	ICOUNT : ICOUNT f_t $-=$ $0.1f_{peak}$	ICOUNT : ICOUNT $f_t = f_{peak}$	$l_1 : t_2$ $f_t = f_{peak}$
$TH1 < s_1 \leq TH2$	ICOUNT : t_1 f_t $+=$ $0.1f_{peak}$	ICOUNT : ICOUNT $f_t = f_{peak}$	ICOUNT : ICOUNT $f_t = f_{peak}$	$l_1 : t_2$ $f_t = f_{peak}$
$TH2 < s_1$	$l_2 : t_1$ $f_t = f_{peak}$	$l_2 : t_1$ $f_t = f_{peak}$	$l_2 : t_1$ $f_t = f_{peak}$	stall : ICOUNT $f_t = f_{peak}$

When DFS is applied to the trailing core and eff_freq is its average operating frequency (normalized to the peak frequency), assuming marginal stalls for the leading thread, the total power in the system is given by the equation:

$$\begin{aligned} DFS_ooo_power = leakage_{leading} + dynamic_{leading} + \\ leakage_{leading} + dynamic_{leading} \times eff_freq/wrongpath_factor \end{aligned} \tag{3.2}$$

If scaling the frequency by a factor eff_freq allows us to scale voltage by a factor $eff_freq \times v_factor$ (in practice, v_factor is greater than 1), the trailing core's power is as shown below.

$$\begin{aligned} DVFS_ooo_power = leakage_{leading} + dynamic_{leading} + \\ leakage_{leading} \times eff_freq \times v_factor + \\ dynamic_{leading} \times eff_freq^3 \times v_factor^2/wrongpath_factor \end{aligned} \tag{3.3}$$

If an in-order core with RVP is employed for the trailing thread, the following equation is applicable, assuming that the in-order core consumes lkg_ratio times less leakage and dyn_ratio times less dynamic power than the out-of-order leading core.

$$\begin{aligned} DFS_inorder_power = leakage_{leading} + dynamic_{leading} + \\ leakage_{leading}/lkg_ratio + RVP_overhead \\ dynamic_{leading} \times eff_freq/(wrongpath_factor \times dyn_ratio) \end{aligned} \tag{3.4}$$

We now consider the effect of parallelizing the verification workload across N in-order trailing cores. Assuming that we only employ DFS for each in-order core, trailing thread power is given by:

$$\begin{aligned} DFS_inorder_WP_power = leakage_{leading} + dynamic_{leading} \\ + N \times leakage_{leading}/lkg_ratio + RVP_overhead \\ N \times dynamic_{leading} \times eff_freq/(wrongpath_factor \times dyn_ratio) \end{aligned} \tag{3.5}$$

Note that the dynamic power remains the same as in Equation (1) – eff_freq goes down by a factor N, but that amount is now expended at N different cores. In other words, the same amount of work is being done in either case. Leakage power increases because leakage is a function of the number of transistors being employed. Parallelization has a benefit only if we are also scaling voltage. Power is then expressed as follows:

$$\begin{aligned} DVFS_inorder_WP_power = v_factor \times leakage_{leading}/lkg_ratio + \\ N \times v_factor^2 \times dynamic_{leading} \times eff_freq/ \\ (wrongpath_factor \times dyn_ratio \times N^2) + RVP_overhead \end{aligned} \tag{3.6}$$

Finally, similar models can be constructed for CRTR and P-CRTR multithreaded models. P_CRTR_ooo represents a model where both trailing threads execute on an SMT out-of-order core, $P_CRTR_inorder$ represents a model where each trailing thread executes on its individual in-order core, and *slowdown* represents the throughput slowdown when executing leading threads together instead of with trailing threads. $RVP_overhead$ includes additional power consumed within the RVQ to enable RVP.

$$Energy_{CRTR} = 2 \times (leakage_{ooo} \ + \ dynamic_{ooo} \times (1 + wrongpath_factor)) \tag{3.7}$$

$$\begin{aligned} Energy_{P_CRTR_ooo} = slowdown \times (2 \times leakage_{ooo} + \\ dynamic_{ooo} \times (wrongpath_factor \ + wrongpath_factor) \\ + dynamic_{ooo} \times eff_freq \times (1 + 1)) \end{aligned} \tag{3.8}$$

$$\begin{aligned} Energy_{P_CRTR_inorder} = slowdown \times (leakage_{ooo} \times \\ (1 + 2/lkg_ratio) \ + dynamic_{ooo} \times (wrongpath_factor + \\ wrongpath_factor) \ + \ 2 \times RVP_overhead + \\ dynamic_{ooo} \times effective_frequency \times (1 + 1)/dyn_ratio) \end{aligned} \tag{3.9}$$

Parameters such as $slowdown$, eff_freq, and $wrongpath_factor$ have to be calculated through detailed simulations. For example, for our simulation parameters, $wrongpath_factor$ was 1.17, $slowdown$ for P_CRTR_ooo was 9.4%, and eff_freq for the single-thread model was 0.44.

3.3.5 Implementation Complexities

This subsection provides an analysis of the complexity introduced by the mechanisms in this paper. Firstly, to enable DFS, each core must have a clock divider to independently control its operating frequency. The clocks may or may not be derived from a single source. The buffers between the two cores must be designed to allow variable frequencies for input and output. Multiple clock domain processors employ such buffers between different clock domains [45]. While changing the frequency of the trailing core, we will make the conservative assumption that the leading and trailing cores are both stalled until the change has stabilized. The dynamic frequency selection mechanism can be easily implemented with a comparator and a few counters that track the size of the RVQ, the current frequency, the interval, and threshold values.

A large slack is required to absorb throughput variations within an interval, also requiring that we implement a large RVQ, LVQ, BOQ, and StB. To accommodate a slack of 1000 instructions, we implement a 600-entry RVQ, a 200-entry BOQ, 400-entry LVQ, and a 200-entry StB per trailing thread. All queues and buffers are modeled as first-in-first-out (FIFO) queues. Their access is off the critical path but can incur nontrivial power overheads. Values are written to and read from these queues in sequential order, allowing them to be implemented with single read and write ports (each row has as many entries as the fetch/commit width). The peak power of the largest structure, the RVQ, was computed to be 0.89 W (with Wattch's power model [46]). It must be noted that low-overhead DFS (such as the single cycle overhead in Montecito) enables a smaller interval, and therefore, smaller slack, RVQ, LVQ, BOQ, and StB. Hence, our assumptions for intercore power overheads are pessimistic.

While an in-order core can yield significant power savings, additional power is expended in implementing RVP. In one possible implementation, each entry of the RVQ now contains the instruction's source operands in addition to the result. This increases the RVQ's peak power consumption from 0.89 W to 2.59 W. As an instruction flows through the in-order pipeline, it reads the corresponding input operands from the RVQ and control signals are set so that the multiplexor before the ALU selects these values as inputs. The pipeline is modified so that the register file is read at the time of commit and not before the execute stage. Our simulations estimate that RVP incurs an additional average power cost of 2.54 W for intercore transfers. This estimate is admittedly simplistic as it does not take the cost of pipeline modifications into account.

3.4 Results

3.4.1 Methodology

We use a multithreaded version of Simplescalar-3.0 [47] for the Alpha AXP ISA for our simulations. The simulator has been extended to implement a CMP architecture, where each core is a two-way SMT processor. Table 3.2 shows relevant simulation parameters. The Wattch [46] power model has been extended to model power consumption for the CMP architecture at 90 nm technology, at 5 GHz, and 1.1 V supply voltage. The aggressive clock gating model (cc3) has been assumed throughout. Wattch's RAM array and wire models were used to compute power dissipated by the intercore buffers (RVQ, LVQ, etc.). The RVQ was modeled as a single-read and single-write ported structure with 150 total rows, each row accommodating results for four instructions (32 bytes of data), resulting in a peak power dissipation of 0.89 W. With RVP included, the size of a row expands to 96 bytes and the peak power dissipation to 2.59 W. To model the intercore bus, power-optimized wires [48] were employed. The distance between two cores was assumed to be equivalent to the width of a single core (4.4mm – obtained from [49] and scaled to a 90 nm process). Assuming a bus width of 35 bytes (without RVP),

Table 3.2. Simplescalar simulation parameters

Branch Predictor	Comb. bimodal/2-level (per core)
Bimodal Predictor Size	16384
Level 1 Predictor	16384 entries, history 12
Level 2 Predictor	16384 entries
BTB	16384 sets, two-way
Branch Mpred Latency	12 cycles
Instruction Fetch Queue	32 (per Core)
Fetch width/speed	4/2 (per Core)
Dispatch/Commit Width	4 (per Core)
IssueQ size	40 (Int) 30 (FP) (per Core)
Reorder Buffer Size	80 (per Thread)
LSQ size	100 (per Core)
Integer ALUs/mult	4/2 (per Core)
FP ALUs/mult	1/1 (per Core)
(Single thread) L1 I-cache	32KB two-way (per Core)
L1 D-cache	32KB two-way, 2-cyc (per Core)
(multithread) L1 I-cache	128KB two-way (per Core)
L1 D-cache	128KB two-way, 2-cyc (per Core)
L2 unified cache	2MB 8-way, 20 cycles (per Core)
Frequency	5 GHz
I and D TLB	256 entries, 8KB page size
Memory Latency	300 cycles for the first chunk

we computed a peak power consumption of 0.67 W for the intercore bus per trailing thread. With RVP, interconnect power consumption is 2 W per thread.

While Wattch provides reliable relative power values for different out-of-order processor configurations, we felt it did not accurately capture the relative power values for out-of-order and in-order cores. An in-order core's power efficiency is derived from its simpler control paths and Wattch primarily models the datapaths within the processor. Hence, the in-order power values obtained from Wattch were multiplied by a scaling factor to bring these numbers more in tune with empirical industrial data. The scaling factor was chosen so that the average power for in-order and out-of-order cores was in the ratio 1:7 or 1:2. This ratio is based on relative power consumptions for commercial implementations of Alpha processors (after scaling for technology) [50]. Our in-order core is modeled loosely after the

quad-issue Alpha EV5 that consumes half the power of the single-thread out-of-order Alpha EV6 and $1/7^{th}$ the power of the multithread out-of-order Alpha EV8. The EV8 is perhaps more aggressive than out-of-order cores of the future (such as Intel's Core) and similarly, the EV5 may be more aggressive than the typical future in-order core (such as Sun's Niagara). Hence, the above ratios are only intended to serve as illustrative design points from a single family of processors. The analytical model can be used to estimate power overheads for other power ratios. The baseline peak frequency for all cores is assumed to be 5 GHz.

As an evaluation workload, we use the 8 integer and 8 floating point benchmark programs from the SPEC2k suite that are compatible with our simulator. The executables were generated with peak optimization flags. The programs were fast-forwarded for 2 billion instructions, executed for 1 million instructions to warm up various structures, and measurements were taken for the next 100 million instructions. To evaluate multithreaded models, we formed a multiprogrammed benchmark set consisting of 10 different pairs of programs. Programs were paired to generate a good mix of high IPC, low IPC, FP, and Integer workloads. Table 3.3 shows our benchmark pairs. Multithreaded workloads are executed until the first thread commits 100 million instructions.

Table 3.3. Benchmark pairs for the multithreaded workloads.

Benchmark Set	**Set #**	**IPC Pairing**
bzip-vortex	1	Int/Int/Low/Low
vpr-gzip	2	Int/Int/High/Low
eon-vpr	3	Int/Int/High/High
swim-lucas	4	FP/FP/Low/Low
art-applu	5	FP/FP/Low/High
mesa-equake	6	FP/FP/High/High
gzip-mgrid	7	Int/FP/Low/Low
bzip-fma3d	8	Int/FP/Low/High
eon-art	9	Int/FP/High/Low
twolf-equake	10	Int/FP/High/High

3.4.2 Single-Thread Results

We begin by examining the behavior of a single-threaded workload. Figure 3.4 shows the IPCs of various configurations, normalized with respect to the IPC of the leading thread executing on an out-of-order core. Such an IPC analysis gives us an estimate of the clock speed that the trailing core can execute at. Since the trailing thread receives load values and branch outcomes from the leading thread, it never experiences cache misses or branch mispredictions. The first bar for each benchmark in Figure 3.4 shows the normalized IPC for such a trailing thread that executes on an out-of-order core with perfect cache and branch predictor. If the trailing thread is further augmented with register value prediction (the second bar), IPC improvement is only minor (for most benchmarks). The third bar shows normalized IPCs for an in-order core with perfect cache and branch predictor. It can be seen that for many programs, the normalized IPC is less than 1.

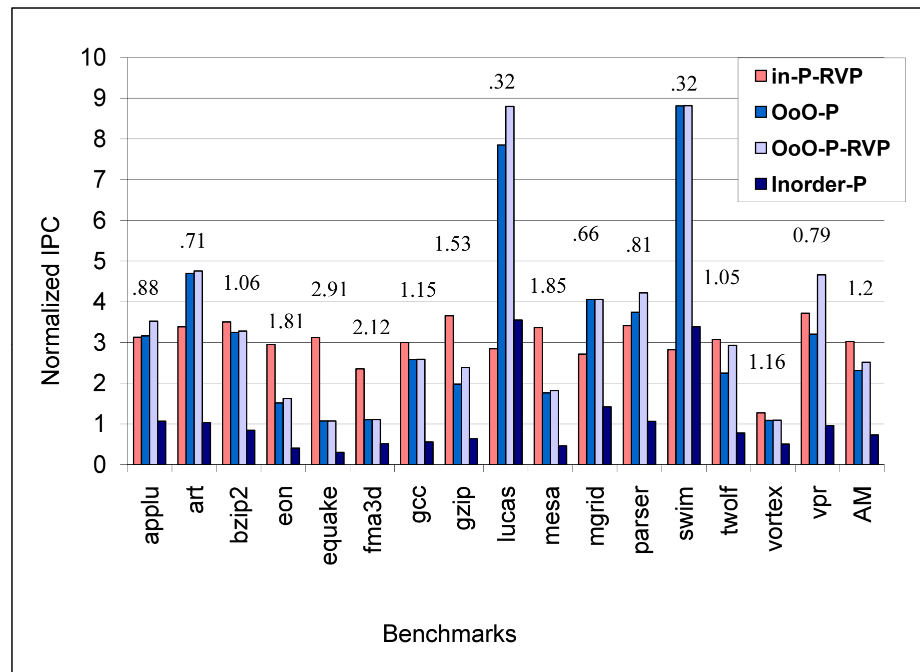


Figure 3.4. IPCs for different models, relative to the baseline out-of-order core. The absolute IPC for the baseline out-of-order core for each program is listed above each set of bars.

This means that an in-order trailing core cannot match the leading thread's throughput unless it operates at a clock speed much higher than that of the out-of-order core. The last bar augments the in-order core's perfect cache and branch predictor with register value prediction (RVP). The increase in IPC is significant, indicating that RVP will be an important feature within an in-order core to allow it to execute at low frequencies. The IPC of a core with perfect cache, branch predictor, and perfect RVP is limited only by functional unit availability and fetch bandwidth. The average normalized IPC is around 3, meaning that, on average, the frequency of a trailing thread can be scaled down by a factor of 3.

We then evaluate the dynamic frequency scaling heuristic on the single-thread programs with different forms of trailing cores. The heuristic is tuned to conservatively select high frequencies so that the leading thread is rarely stalled because of a full RVQ. The performance loss, relative to a baseline core with no redundancy, is therefore negligible. The second bar in Figure 3.5 shows power consumed by a trailing out-of-order core (with perfect cache and branch predictor) that has been dynamically frequency scaled.

Such a trailing redundant core imposes a power overhead that equals 53% of the power consumed by the leading core. Without the DFS heuristic, the trailing core would have imposed a power overhead of 90% (first bar in Figure 3.5). On average, the trailing core operates at a frequency that is 0.44 times the frequency of the leading thread. Note that DFS does not impact leakage power dissipation. The net outcome of the above analysis is that low trailer frequencies selected by the DFS heuristic can reduce the trailer core's power by 42% and the overall processor power (leading and trailing core combined) by 22%. Figure 3.5 also shows the power effect of employing an in-order trailing core. Future CMPs will likely be heterogeneous, providing a good mix of out-of-order and in-order cores [50]. As seen previously, a perfect cache and branch predictor is not enough to allow the in-order core's IPC to match the leading core's IPC. Hence, the system has been augmented with register value prediction. We have pessimistically assumed that the buffers between the two cores now carry two additional 64-bit values per instruction, leading to an

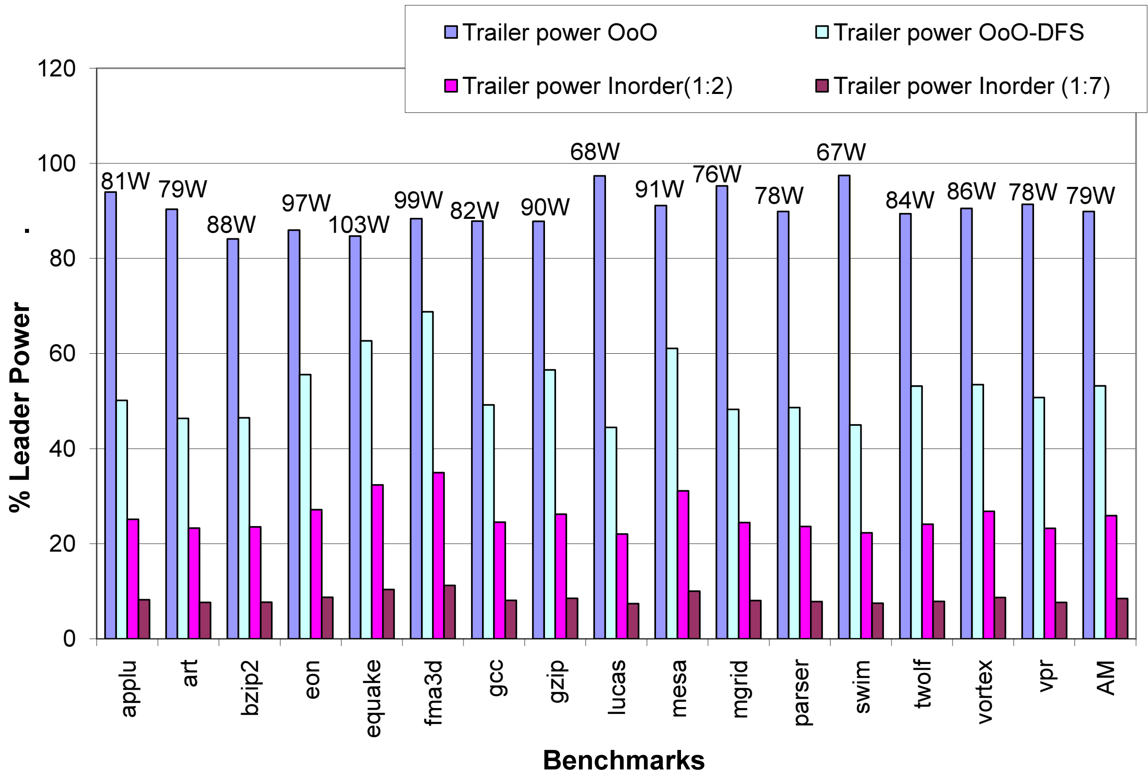


Figure 3.5. Power consumed by the trailing core as a function of the power consumed by the leading core. The number above each set of bars represents the absolute value of power dissipated by the leading core for each benchmark.

additional average power dissipation of 2.54 W. With the in-order to OoO power ratio of 1:2, we observe that the redundancy mechanism now consumes less than 26% of the power consumed by the leading thread. The frequency selected by the DFS heuristic for the in-order core is on average 0.42 times that of the leading core's frequency. For the in-order to OoO power ratio of 1:7, the power consumed by the trailing thread is 8.5% of the leading thread.

For all the above simulations, we assume an interval length of 1000 cycles, when making frequency decisions. Frequency changes were made for 70% of all intervals. If we assume that a frequency change stalls the processor for 10 (peak frequency) cycles, the total overhead is only 0.7%. Our frequency change overhead is very conservative when compared to the recent implementation of DFS in Intel's Montecito core, where a frequency change is effected in a single cycle. The frequency change overhead can also be reduced by incorporating hysteresis within the algorithm, but

this occasionally leads to increased slack and stalls for the leading thread. Given the low 0.7% performance overhead, we chose to not include hysteresis and instead react quickly to variations in slack.

Based on the above results, we make the following general conclusions. Executing the trailing thread on an out-of-order core has significant power overheads even if the trailing core's frequency is scaled (partially because DFS does not impact leakage). An in-order core has much lower power overheads, but poor IPC characteristics, requiring that it operate at a clock speed higher than the leading core. The IPC of the in-order core can be boosted by employing register value prediction. This requires us to invest about 2.54 W more power in transmitting additional data between cores (a pessimistic estimate), but allows us to operate the in-order core at a frequency that is less than half the leading core's peak frequency. Hence, this is a worthwhile trade-off, assuming that the dynamic power consumed by the in-order core is at least 5 W.

3.4.3 Multithread Workloads

Next, we examine the most efficient way to execute a multithreaded workload. As a baseline, we employ the CRTR model proposed by Gomaa et al. [35], where each out-of-order core executes a leading thread and an unrelated trailing thread in SMT fashion. Within the power-efficient P-CRTR-OoO, both leading threads execute on a single SMT out-of-order core and both trailing threads execute on a neighboring SMT out-of-order core that can be frequency scaled. The last bar in Figure 3.6 shows the total leading thread throughput for CRTR for each set of program pairs defined in Table 3.3. The first bar shows the total leading thread throughput in a baseline system where both leading threads execute on a single SMT out-of-order core (no redundant threads are executed). It can be seen that the throughput of CRTR is about 9.7% better than a system where two leading threads execute on the same out-of-order core. This is because each leading thread in CRTR is co-scheduled with a trailing thread that does not execute wrong-path instructions and poses fewer conflicts for resources (ALUs, branch predictor, data

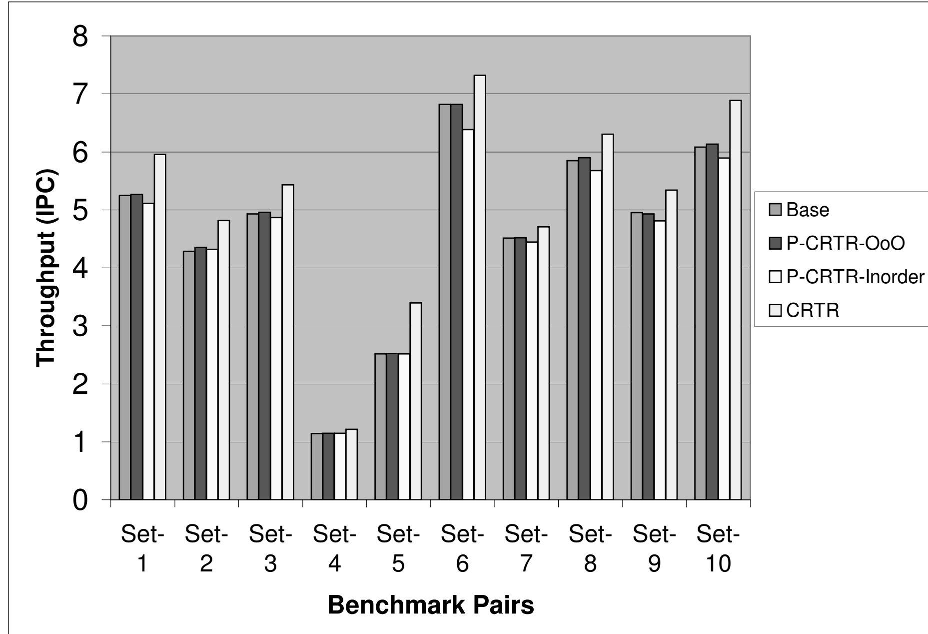


Figure 3.6. Total IPC throughput for multithreaded workloads.

cache, etc.). The second bar in Figure 3.6 shows IPCs for P-CRTR-OoO. The DFS heuristic selects frequencies such that the leading core is rarely stalled and throughputs are very similar to that of the baseline system, about 9.4% lower than CRTR, on average. The results of the earlier subsection indicate that an in-order core augmented with RVP is likely to entail a lower power overhead. Hence, we also evaluate a system (P-CRTR-inorder), where two leading threads execute on an SMT out-of-order core and the two trailing threads execute (by themselves) on two in-order cores with RVP and DFS. Again, the throughput of P-CRTR-inorder is similar to that of the baseline system, about 12% lower than CRTR, on average. Note, however, that P-CRTR-inorder is likely to have a lower area overhead than the other organizations in Figure 3.6 because the area occupied by two single-threaded in-order cores is less than the area occupied by one SMT out-of-order core [50]. The performance penalty for P-CRTR can be primarily attributed to higher ALU, cache, and branch predictor contention. For example, the average L1 cache miss rate for leading threads in P-CRTR was 6.5% higher than that in CRTR.

Figure 3.7 shows the $Energy \times Delay^2$ (ED^2) metric for different forms of P-CRTR, normalized to that for CRTR. Figures 3.6 and 3.7 provide the data necessary to allow readers to compute other metrics in the $E - D$ space. The first bar shows ED^2 for P-CRTR-OoO, where both trailers execute on an SMT out-of-order core. The DFS heuristic scales frequencies for the trailing core, allowing it to consume less total power than CRTR. However, CRTR has a throughput advantage that allows it to have a better (lower) ED^2 than P-CRTR for many programs. On average ED^2 of P-CRTR is 17% more than CRTR. While CRTR imposes an average power overhead of 97% for redundancy, P-CRTR imposes an overhead of 75%. The effective frequency for the trailing core is much higher (0.77 times peak frequency) than that seen for the single-thread workloads because the selected frequency has to be high enough to allow both threads to match leading thread throughputs. Some workloads (sets 4 and 7) are able to lower their trailer frequencies enough to yield lower ED^2 than CRTR. This was also observed for other workloads that had a similar combination of Int/FP/IPC.

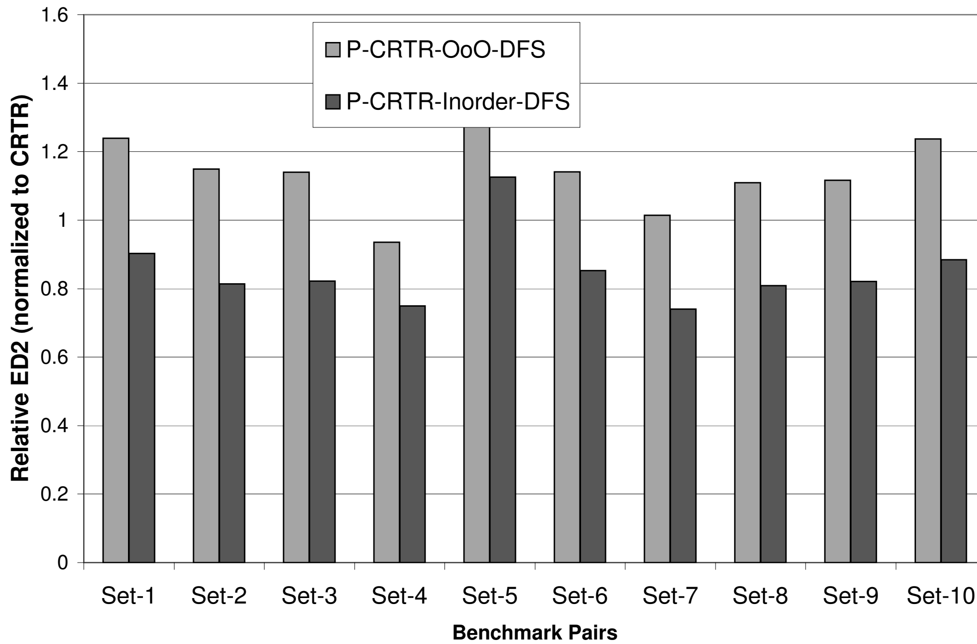


Figure 3.7. ED^2 for the entire system for various forms of trailers, normalized to the ED^2 of CRTR.

In general, workloads composed of low IPC programs (those with high branch mispredict rates and cache miss rates) are likely to see a higher benefit from perfect cache and branch predictor, leading to low trailer frequencies and better overall ED^2 with P-CRTR. When scheduling redundant threads on a CMP of SMT out-of-order cores, the operating system can optimize ED^2 by taking program behavior into account and accordingly adopting a schedule similar to CRTR or P-CRTR.

The second bar in Figure 3.7 represents the P-CRTR-inorder model. We assume that the in-order to out-of-order power consumption ratio is 1:7 for this graph. By executing the trailing thread on a frequency-scaled in-order core with perfect cache, branch predictor, and RVP, significant power reductions are observed (enough to offset the additional power overhead of data transfers between cores). On average, P-CRTR-inorder improves ED^2 by 15%, relative to CRTR. The average power overhead of redundancy is 20%. Benchmark set-5 has 12% higher ED^2 when compared to CRTR due to 25% performance loss. We observed that by co-scheduling leading threads on the same core, branch predictor conflicts increased signficantly for this benchmark pair. The total power overhead associated with RVP for multithreaded workloads is 5.64 W.

The net conclusion is similar to that in the previous subsection. Leading threads executing on an out-of-order core (in single- or multithreaded mode) can be verified efficiently on in-order cores. While more data have to be transmitted between cores, the power-efficiency of an in-order core compensates for the data transfer overhead.

3.4.4 Potential for Voltage Scaling

Frequency scaling is a low-overhead technique that trades off performance and power and allows us to reduce the dynamic power consumed by the trailing thread. One of the most effective techniques to reduce power for a minor performance penalty is dynamic voltage and frequency scaling (DVFS). If our heuristic determines that the trailer can operate at a frequency that is half the peak frequency (say), it may be possible to reduce the voltage by (say) 25% and observe dynamic power reduction within the trailer of 72%, instead of the 50% possible with just

DFS. While DFS does not impact leakage power, DVFS can also reduce leakage as (to a first order) leakage is linearly proportional to supply voltage [51]. DVFS can be combined with body-biasing to further reduce leakage power [52]. However, these techniques require voltage changes that can consume a large number of cycles, of the order of $50\mu s$ [53]. Even if voltage is modified only in small steps, each voltage change will require tens of thousands of cycles. If an increase in frequency is warranted, the frequency increase cannot happen until the voltage is increased, thereby causing stalls for leading threads. As observed earlier, a frequency change is made at the end of 70% of all 1000-cycle intervals. It is difficult to design a DFS mechanism that increases frequency only once every 100,000 cycles on average, and poses minimal stalls for the leading thread. Therefore, it is unlikely that the overhead of dynamic voltage scaling will be tolerable.

We however observed that there may be the potential to effect some degree of conservative voltage scaling. Figure 3.8 shows a histogram of the percentage of intervals spent at each frequency by the in-order trailer with RVP. Peak frequency is exercised for only 0.56% of all intervals. If we operate at a low voltage that can support a frequency of 0.9× peak frequency, but not the peak frequency, we will be forced to increase voltage (and stall the leader for at least 10,000 cycles) for a maximum of 0.56% of all 1000-cycle intervals. This amounts to a performance overhead of up to 5.6%, which may be tolerable. The corresponding power benefit may be marginal, especially considering the small amount of power consumed within each in-order core, as illustrated below.

Consider the following example scenario when the trailer is executed on a frequency-scaled in-order core augmented with RVP. If 100 units of power are consumed within the leader, an in-order trailer will consume 14 units of power (ratio similar to the EV5 and EV8), of which about 10 units can be attributed to dynamic power. A DFS heuristic with an effective frequency of 0.5 will reduce the in-order core's dynamic power by 5 units. Thus, out of the total 109 units consumed by this system, 4 units can be attributed to in-order leakage and 5 units

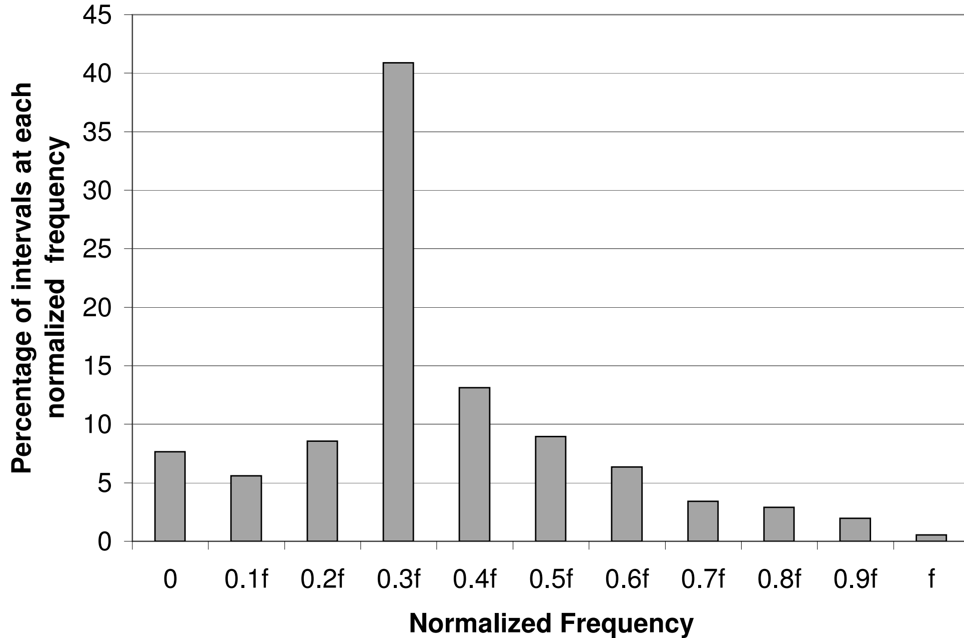


Figure 3.8. Histogram showing percentage of intervals at each normalized frequency.

to in-order dynamic power. Any additional optimizations to this system must take note of the fact that the margin for improvement is very small.

Based on the simple analysis above, we also examine the potential benefits of parallelizing the verification workload [37]. With RVP, the trailing thread has a very high degree of ILP as every instruction is independent. Instead of executing a single trailing thread on an in-order core, the trailing thread can be decomposed into (say) two threads and made to execute in parallel on two in-order cores. When the workload is parallelized by a factor of 2, the effective frequency can be lowered by a factor of 2. Hence, the power consumed by this system will equal 113 units (100 for leading core + 8 for leakage on two in-order cores + 2.5 for dynamic on first in-order core + 2.5 for dynamic on second in-order core). Thus, parallelization with DFS does not reduce dynamic power, but increases leakage power and is therefore not worthwhile. For parallelization to be effective, DFS has to be combined with a technique such as DVFS or body-biasing. Assume that an effective frequency of 0.5 can be combined with a voltage reduction of 25% (similar to that in the Xscale). Parallelization on two in-order cores yields a total

power consumption of 108.8 units (100 for leading core + 6 for leakage which is a linear function of supply voltage + 1.4 for dynamic on first in-order core which is a quadratic function of supply voltage + 1.4 for dynamic on second in-order core). The reduction in dynamic power is almost entirely negated by the increase in leakage power. Clearly, different assumptions on voltage and frequency scaling factors, leakage, in-order power consumption, etc., can yield different quantitative numbers. In the next subsection, we use our analytical model to show that for most reasonable parameters, workload parallelization yields little power benefit even when we aggressively assume that voltage scaling has no overhead.

There are other problems associated with voltage scaling: (i) Lower voltages can increase a processor's susceptibility to faults. (ii) As voltage levels and the gap between supply and threshold voltages reduce, opportunities for voltage scaling may cease to exist. (iii) Parallelization has low scalability in voltage terms – parallelizing the workload across four cores allows frequency to be scaled down by a factor of four, but reductions in voltage become increasingly marginal.

3.4.5 Sensitivity Analysis

As an example application of the analytical model, we present power overheads as a function of the contribution of leakage to the baseline out-of-order leading core (Figure 3.9). The three forms of trailers shown are an out-of-order core with the DFS heuristic, and in-order cores with RVP and DFS, that consume 0.55 times and 0.14 times the power of the out-of-order core. The overall conclusions of our study hold for all of these design points.

We discussed workload parallelization in the earlier subsection and re-confirm our observations with the help of analytical models for various design points. Figure 3.10 shows the effect of workload parallelization and leakage contribution on the trailing core's power, where the trailing thread executes on an in-order processor enabled with DVFS. Based on detailed simulation results, we assume $wrongpath_factor = 1.17$, $eff_freq = 0.44$, $v_factor = 1.25$, lkg_ratio and dyn_ratio = 1.8. For N = 2, as the contribution of leakage power increases, workload

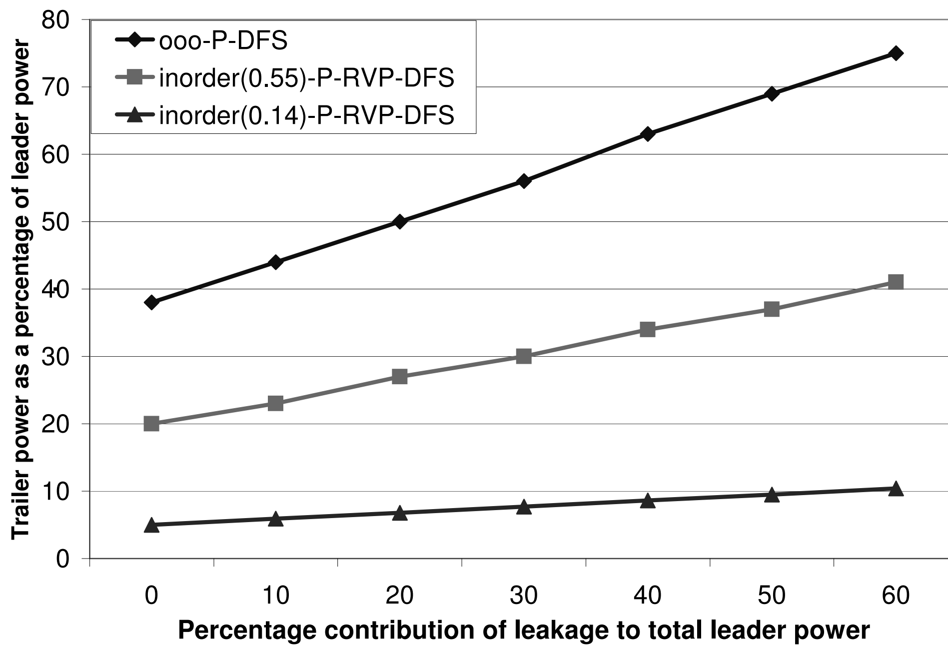


Figure 3.9. Trailer power as a function of contribution of leakage to the baseline processor.

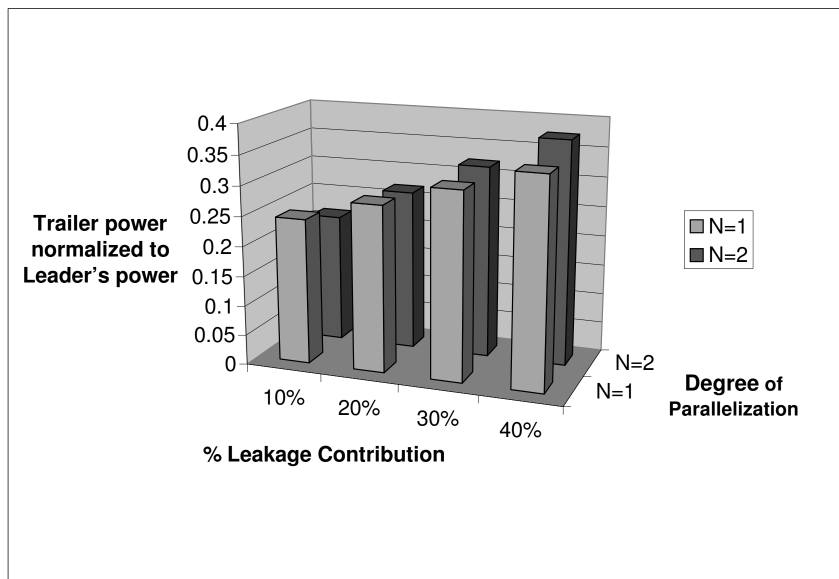


Figure 3.10. Power overhead of the trailing core, relative to the leading core, with and without parallelizing the verification workload.

parallelization yields marginal improvement over the base case (N = 1). Note that the base case has a single DFS enabled in-order core and does not employ voltage scaling. Even for low leakage contribution, the power reduction with workload parallelization is only 2.5%.

The effect of various parameters on IPC is harder to capture with analytical models and we report on some of our salient observations below. (i) The relative benefit of a perfect cache and branch predictor are significant for most processor models. For example, increasing the window size improves the ability of the baseline out-of-order core to tolerate cache misses, and likewise, also improves the ability of the core with the perfect cache/bpred to mine greater ILP. (ii) We have observed that for a multithreaded workload, scheduling a leading and trailing thread on the same SMT core (as in CRTR) yields a 9.4% throughput improvement over a model where the leading threads are scheduled on the same SMT core. As the total number of functional units is increased, this performance gap reduces to 6.3% because contention for resources becomes less of an issue. (iii) For an in-order core with RVP, the only limitation to IPC is the number of functional units available and the fetch bandwidth. The effective frequency of the in-order core can be reduced by increasing the number of functional units and hence, the IPC. Likewise, the ability to fetch from multiple basic blocks in the same cycle has a much greater impact on the IPC of the in-order core with RVP, than on other cores. (iv) The slack between leading and trailing threads is closely related with interval size. If we examine slack every 1000 cycles to make a decision for trailer frequency, the slack must be about 1000 in order to absorb a significant IPC change of 1.0 in the leading thread in the next interval. A smaller slack can lead to the intercore buffers getting full and stalling the leading thread. A large slack allows more opportunities for frequency reduction, but incurs a nontrivial power overhead for the intercore buffers.

3.5 Related Work

Many fault-tolerant architectures [34–36, 38, 40, 54–56] have been proposed over the last few years and our baseline models are based on previously proposed redun-

dant multithreading designs. Most of this prior work has leveraged information produced by the leading thread, but the focus has been on the performance-reliability trade-off with few explicit proposals for power-efficiency. AR-SMT [55] was the first design to use multithreading for fault detection. AR-SMT proposed sending all register values to the trailing thread to boost its performance. In our work, we exploit register values to enable in-order execution of the trailing thread for power-efficiency. Mukherjee et al. later proposed fault detection using simultaneous multithreading and chip-level redundant multithreading [36, 38]. Vijaykumar et al. augmented the above techniques with recovery mechanisms [35, 40]. Most of these research efforts have been targeted at improving thread-level throughput and have not been optimized for power-efficiency. Gomaa et al. [35] discuss techniques, such as Death and Dependence Based Checking Elision (DDBCE), to reduce the bandwidth requirements (and hence, power overheads) of the intercore interconnect. Our proposals, on the other hand, advocate transferring more data between threads to enable power optimizations at the trailer. Mukherjee et al. [57] characterize the architectural vulnerability factors (AVF) of various processor structures. Power overheads of redundancy can be controlled by only targeting those processor structures that have a high AVF.

Some designs, such as DIVA [34] and SHREC [39], are inherently power-efficient because the helper pipelines they employ to execute redundant instructions are in-order-like. DIVA has two in-order pipelines – *Checkcomm* that checks all memory values and *Checkcomp* that checks all computations. These helper pipelines are fed with input values generated by the primary pipeline. However, these designs require (potentially intrusive) modifications to the pipeline of the conventional primary microarchitecture and the helper pipelines cannot be used to execute primary threads. We extend this concept by executing the redundant thread on a general-purpose in-order core augmented with register value prediction and by limiting the data that has to be extracted from the primary pipeline. Even though DIVA was among the first RMT proposals, recent research in this area has moved away from that concept and focused more on the exploitation of heavily-threaded

hardware. Our conclusions show that the DIVA concepts are worth re-visiting and are complexity-effective in a processor with heterogeneous in-order and out-of-order cores.

A recent paper by Rashid et al. [37] represents one of the first efforts at explicitly reducing the power overhead of redundancy. In their proposal, the leading thread is analyzed and decomposed into two parallel verification threads that execute on two other out-of-order cores. Parallelization and the prefetch effect of the leading thread allow the redundant cores to operate at half the peak frequency and lower supply voltage and still match the leading thread's throughput. Our approach differs from that work in several ways: (i) In [37], redundant loads retrieve data from caches, leading to complex operations to maintain data coherence between multiple threads. In our implementation, redundant instructions receive load results and even input operands from the leading thread. We claim that by investing in intercore bandwidth, trailer core overheads can be reduced. (ii) [37] relies on voltage scaling and body-biasing to benefit from parallelization. We have explicitly steered away from such techniques because of the associated overheads. Unlike their work, we leverage dynamic frequency scaling and in-order execution. (iii) Our analytical results show that parallelization of the verification workload yields little benefit if we are already employing in-order cores augmented with RVP.

Some papers have looked at reducing resource and instruction redundancy for reducing performance and power overheads in RMT techniques without reducing the fault coverage. Kumar and Aggarwal [58] apply register re-use and narrow-width operand register sharing techniques to reduce the performance and power overheads in the register file. In another recent paper by the same authors [59], many instructions are classified as *self-checking* such as those that have a *zero* operand. The results produced by these instructions will often be equal to their nonzero operands and these instructions need not be redundantly executed by the trailing thread. These techniques indirectly reduce the power overheads by executing fewer instructions for verification. Other approaches that reduce the power overheads indirectly are RMT techniques that reduce the fault coverage

such as opportunistic RMT [41]. In that work, in addition to exploiting instruction reuse, redundant threads execute only when the primary thread is stalled on L2 cache misses. This paper explores techniques that can reduce power and area overheads while maintaining a constant level of error coverage. These techniques are often orthogonal to other techniques that, for example, trade off reliability for better performance or power.

3.6 Summary

In this chapter, we have presented novel micro-architectural techniques for reducing the power overheads of redundant multithreading. When executing leading and trailing redundant threads, we take advantage of the fact that the leading thread prefetches data and resolves branches for the trailing thread. The results of the leading thread also allow the trailing core to implement a perfect register value predictor. All this information from the leading thread makes it possible for the trailing thread to achieve high IPC rates even with an in-order core, thereby justifying the cost of high intercore traffic. Dynamic frequency scaling further helps reduce the power consumption of the trailing thread. Our results indicate that workload parallelization and voltage scaling hold little promise. We quantify the power-performance trade-off when scheduling the redundant threads of a multithreaded workload and derive analytical models to capture the insight from our detailed simulations. None of the mechanisms proposed in this work compromise the error coverage of the baseline system. We will demonstrate next on how we will utilize this power-efficient redundantly threaded architecture for 3D stacking. We also show how the power reduction techniques proposed in this chapter help reduce the thermal overheads in 3D stacking.

CHAPTER 4

LEVERAGING 3D TECHNOLOGY FOR RELIABILITY

In this chapter, we focus on the first major component of this thesis that investigates the use of 3D technology for designing reliable processors. We outline a design that represents a complexity-effective solution to the problem of comprehensive error detection and recovery. This solution builds upon the power-efficient reliable processor architecture proposed in the previous Chapter 3. We then implement this reliable processor model in 3D technology by stacking the checker on the second die to leverage the snap-on functionality. We comprehensively evaluate design choices for this second die, including the effects of L2 cache organization, deep pipelining, and frequency. We demonstrate that 3D is an attractive option as it lets us implement the checker core die in an older process technology that is more error-resilient and also has lower leakage power. The proposed techniques in this chapter showcase reliability as the unexplored and especially compelling application of die heterogeneity.

This chapter is organized as follows. In Section 4.2, we first describe our checker processor model. Section 4.3 evaluates the impact of leveraging 3D interconnects for intercore communication. We also quantify the impact of this approach on temperature, interconnect length, and error tolerance. The use of multiple dies enables the use of different parameters for the primary and checker cores. Section 4.4 analyzes the effect of older technology parameters for the checker on overall power, temperature, and error rates. We discuss related work in Section 4.5 and summarize our conclusions in Section 4.6.

4.1 3D Technology for Reliability

3D technology is emerging as an intriguing prospect for the future (see [12] for a good overview). This technology enables the vertical stacking of dies with low-latency communication links between dies. 3D stacking offers three primary advantages that make it a compelling research direction:

- Stacking of heterogeneous dies: A concrete application of this approach is the 3D stacking of DRAM chips upon large-scale CMPs. interdie vias can take advantage of the entire die surface area to implement a high bandwidth link to DRAM, thereby addressing a key bottleneck in CMPs that incorporate nearly a hundred cores [13, 15].

- “Snap-on” analysis engines: Chips employed by application developers can be fitted with additional stacked dies that contain units to monitor hardware activity and aid in debugging [16]. Chips employed by application users will not incorporate such functionality, thereby lowering the cost for these systems.

- Improvement in CPU performance/power: The components of a CPU (cores, cache banks, pipeline stages, individual circuits) can be implemented across multiple dies. By lowering the penalties imposed by long wires, performance and power improvements are possible.

All of the above advantages of 3D can be exploited if we implement the checker core on a die placed above the CPU die: (i) The communication of results between cores is very efficient in terms of delay and power as short interdie wires can be used. By isolating the checker core to a separate die, the layout and wiring of the main CPU die can be simplified. (ii) CPU dies and checker dies can be independently fabricated and chip manufacturers can offer products with a varying number of checker dies. (iii) The checker cores can be implemented in a process technology that is more resilient to soft errors and dynamic timing errors.

The one salient disadvantage of 3D technology is that it increases on-chip power density and temperature. Temperature increases can have a significant impact on lifetime reliability, soft error rates, leakage power dissipation, dynamic timing error

rates, and performance. We consider the above effects and attempt to alleviate them with older process technologies, deeper pipelines, and three-dimensional L2 cache organizations. An evaluation of die yield is beyond the scope of this study. While the fabrication of small-sized dies for a 3D chip will improve yield, this advantage may be offset by complications in the 3D manufacturing process. The effect on yield will become clearer as 3D technology matures.

The major contributions of the work in this chapter are as follows:

- We quantify the impact of a 3D checker core on power, temperature, performance, and interconnect overhead.

- We argue for the use of an older process technology for the checker die to improve its error resilience and quantify the impact of this choice on power, temperature, and performance.

- We show that a high-ILP checker can operate at low frequencies, allowing much more timing slack in each pipeline stage and greater error resilience.

- We argue against the use of deep pipelines for the checker core.

4.2 Baseline Reliable Processor Model

This section describes the implementation of a checker core that redundantly executes a copy of an application thread. This implementation is based on the architecture described in the previous Chapter 3 and is capable of detecting and recovering from soft, hard, and dynamic timing errors. We are not claiming that this is an optimal design for comprehensive error coverage, a topic that remains an active area for research. We are simply postulating this design as a potential implementation that can be employed for the rest of our 3D layout analysis.

We briefly recap the salient features of the reliable architecture. The architecture consists of two communicating cores that execute copies of the same application for fault-detection (a flavor of *chip-level redundant multithreading (CRT)*). One of the cores (the trailing core) executes behind the second core (the leading core) by a certain amount of slack to enable checking for errors. The leading core

communicates its committed register results to the trailing core for comparison of values to detect faults (the baseline reliable processor architecture . The leading core also communicates branch outcome and load values to the trailing core. We take advantage of this performance optimization and employ in-order execution and dynamic frequency scaling for reducing power overheads of the checker core. As discussed in the previous chapter, RVP is also employed to extract the high ILP of an in-order checker core.

It must be noted that such an efficient checker core is facilitated by investing in a large amount of intercore traffic (register operands, load values, and branch outcomes are transmitted to the trailing core). This is worthwhile even in a 2D layout because of the significant energy savings it enables in the checker core. The intercore traffic is an especially worthwhile investment in a 3D layout where communication between dies happens on low-latency and low-power vias.

4.3 Proposed 3D Implementation

In a 2D layout of the reliable processor described in Section 4.2, the out-of-order leading core and in-order checker core are located next to each other, as shown in Figure 4.1(a). Intercore communication of results is implemented on interconnects that are routed over a number of other blocks on higher metal layers. These are typically long pipelined wires with a few repeaters and latches per wire that consume nontrivial amounts of power. These repeaters and latches also require silicon area in the blocks that the interconnects happen to fly over. As a result of these wiring and silicon needs of the checker, the layout of the leading core will be negatively impacted. The integration of such an invasive checker core on a 2D die will likely increase the area, wiring complexity, and cycle time for the leading core.

In a 3D die-stacked chip, two separately manufactured dies can be bonded together in various ways. In this work, we focus on Face-to-Face (F2F) bonding of two dies, as shown in Figure 4.1(b). Intra-die communication happens on conventional interconnects implemented on that die's metal layers. Interdie communication happens on interdie (or die-to-die) vias that are simply an extension of the vias

used to access each die's metal stack. A collection of these interdie vias is referred to as an "interdie via pillar" [30]. Thus, a register value in the lower die can now be driven vertically on an interdie via pillar to a metal layer on the die above and then routed horizontally to the unit on the upper die that needs the value. With an appropriate layout, the horizontal traversal on the upper die can be made short. With such an organization, the primary requirement of the intercore interconnects is metal space to implement the interdie via pillars. The isolation of the checker core and its associated logic and wiring to a separate die minimizes the impact on the leading core's floorplan, wiring, and cycle time. The lower die can therefore also be sold as a stand-alone entity that offers reduced reliability (and nearly the same performance and power as a 2D processor without the pillars). For customers that require high reliability, the lower die is combined with a corresponding upper die that incorporates a checker core to monitor the register values produced on the interdie pillars (shown in Figure 4.1(c)). This "snap-on" functionality of 3D dies was recently proposed by Mysore et al. [16] to implement software profiling and debugging tools. This work proposes a snap-on 3D checker core and various aspects of such an organization are discussed next. The next section evaluates the use of an older process technology to implement the checker core. It must be noted that a primary benefit of the proposed 3D organization is the minimal impact on the layout, wiring, and cycle time of the leading core; quantifying this "noninvasive" effect of the 3D checker is beyond the scope of this paper.

4.3.1 Methodology

Our thermal model is based on the Hotspot-3.1 [60] grid model. The modeling parameters and thermal constants for each layer are as described in [20, 22]. The heat sink is placed close to the bottom die that includes the high-power out-of-order leading core. The leading core's floorplan is based on the Alpha EV7. The Wattch [46] power model has been extended to model power consumption for the processor at 65 nm technology, at 2 GHz, and 1 V supply voltage. We have added gate capacitance and metal capacitance scaling factors to scale Wattch's 90 nm

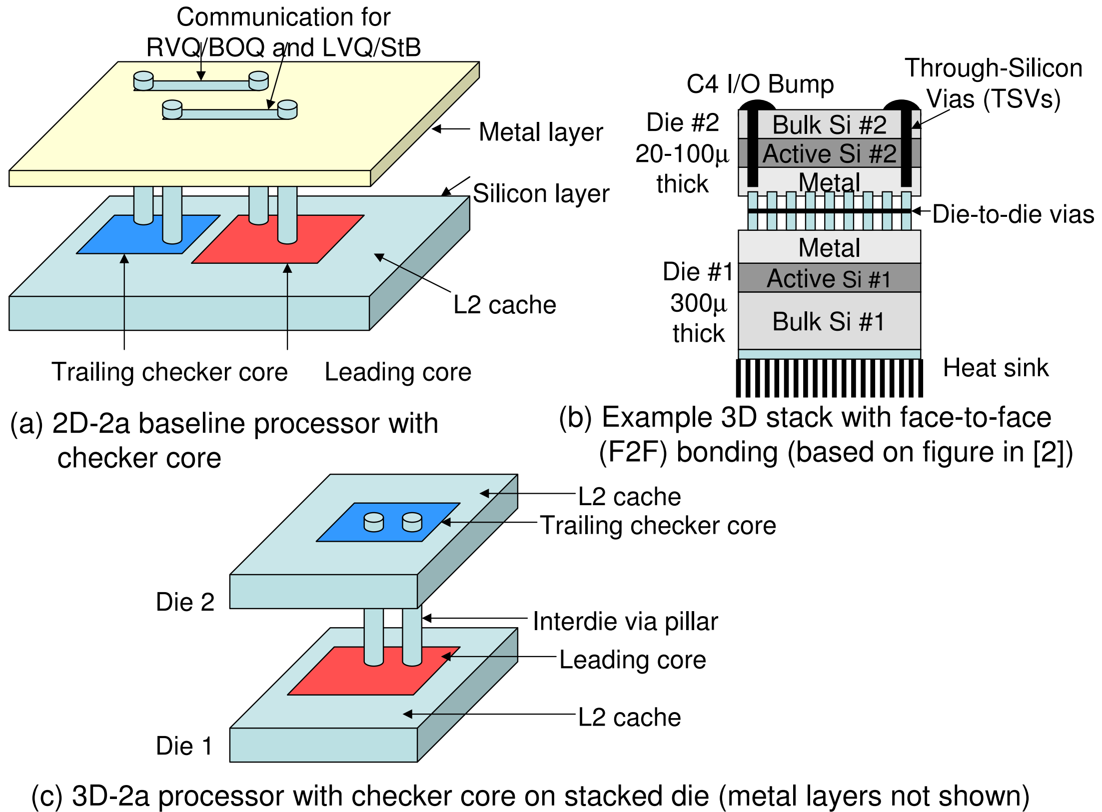


Figure 4.1. Baseline (a) and proposed (c) processor models and example F2F 3D stack (b). Figure not drawn to scale.

model to 65 nm as done in Orion [61]. The aggressive clock gating model (cc3) has been assumed throughout and a turn-off factor of 0.2 has been used to account for higher leakage power in a 65 nm process. The power and area values for the L2 cache's network routers have been derived from Orion [61]. These power values are fed to the Hotspot thermal model to estimate temperatures. For our performance simulations, we employ a multithreaded version of Simplescalar-3.0 [47] for the Alpha AXP ISA.

For our cache power, delay, and area models, we employ CACTI-4.0 [62]. For large L2 cache organizations with nonuniform cache access (NUCA), we employ the methodology in [4] to compute delay and power per access. For example, when modeling a large 15 MB L2 cache, the cache is partitioned into 15 1 MB banks. The banks are accessed via a grid network where each hop consumes four cycles

(one cycle for the link latency and three cycles for the router latency). The link latency is computed based on wire delay speeds for top level metal [4] and the router latency is typical of conventional four-stage routers where the switch and virtual channel allocator stages execute in parallel [63]. The area of the router is taken into account in estimating the area of each cache bank. The size of the large cache is in keeping with modern trends (the Intel Montecito has a 12 MB L3 cache for each core [43]). The cache partitioning not only allows improved performance from nonuniform access, it also allows us to leverage additional cache banks on the second die in a seamless manner. Our baseline single-die processor has 6 1 MB banks surrounding the core and operates as a six-way 6 MB L2 cache. This model is referred to as "2d-a" throughout the chapter, where "a" refers to the total relative area. When the second die is added, the spare area on the second die allows us to implement nine additional 1 MB banks, modeled as a total fifteen-way 15 MB L2 cache. This model is referred to as "3d-2a" because it has twice the total area of the "2d-a" baseline. For comparison purposes, we also model a 2D die that incorporates a checker core and a fifteen-way 15 MB L2 cache, thus providing as many transistors as the 3D chip and referred to as "2d-2a". It must be noted that the 2d-2a model has a larger surface area and hence a larger heat sink than the other two models. All relevant simulation parameters are summarized in Tables 4.1, 4.2, and 4.3.

Table 4.1. Thermal model parameters

Parameter	Value
Bulk Si Thickness die1(next to heatsink)	750μm
Bulk Si Thickness die2 (stacked die)	20μm
Active Layer Thickness	1μm
Cu Metal Layer Thickness	12μm
D2D via Thickness	10μm
Si Resistivity	0.01 $(mK)/W$
Cu Resistivity	0.0833$(mK)/W$
D2D via Resistivity (accounts for air cavities and die to die interconnect density)	0.0166 $(mK)/W$
HotSpot Grid Resolution	50x50
Ambient temperature	47 ℃

Table 4.2. Area and power values for various blocks

Block	Area	Average Power
Leading Core	19.6 mm^2	35 W
In-order Core	5 mm^2	7 W / 15 W
1MB L2 Cache Bank	5 mm^2	0.732 W dynamic per access and 0.376 W static power
Network Router	0.22 mm^2	0.296 W

Table 4.3. Simplescalar simulation parameters

Branch Predictor	Comb. bimodal/2-level (per core)
Bimodal Predictor Size	16384
Level 1 Predictor	16384 entries, history 12
Level 2 Predictor	16384 entries
BTB	16384 sets, 2-way
Branch Mpred Latency	12 cycles
Instruction Fetch Queue	32
Fetch width/speed	4/1
Dispatch/Commit Width	4
IssueQ size	20 (Int) 15 (FP)
Reorder Buffer Size	80
LSQ size	40
Integer ALUs/mult	4/2
FP ALUs/mult	1/1
(Single thread) L1 I-cache	32KB 2-way
L1 D-cache	32KB 2-way, 2-cyc
L2 NUCA cache	6MB 6-way
Frequency	2 GHz
I and D TLB	256 entries, 8KB page size
Memory Latency	300 cycles for the first chunk

As an evaluation workload, we use the 7 integer and 12 floating point benchmark programs from the SPEC2k suite that are compatible with our simulator. The executables were generated with peak optimization flags. The programs are simulated over 100 million instruction windows identified by the Simpoint framework [64].

We model two different policies for L2 NUCA cache access. In the first approach, we distribute the sets across cache banks; given an address, the request is routed to a unique cache bank. While this approach is simpler, it causes all banks to be uniformly accessed, potentially resulting in long average cache access times. In a second approach, we distribute the ways across cache banks. Since a block can now

reside anywhere, we need a mechanism to search for the data. Instead of sending the request to multiple banks, we maintain a centralized tag array near the L2 cache controller. The tags are first looked up before forwarding the request to the corresponding L2 cache bank. With this policy, one of the L2 cache banks in the floorplan is replaced by this centralized tag structure. Depending on the approach, an increase in cache size is modeled as an increase in the number of sets or ways. We adopt the distributed-sets policy for most of the evaluations in this work.

4.3.2 Thermal Overheads of 3D Checkers

We first evaluate the thermal impact of implementing a checker core on a separate die. Wattch does not provide accurate relative power estimates for in-order and out-of-order cores as it does not model control paths. There is also a wide spectrum in power and area characteristics of commercial implementations of in-order cores. For example, each core in the Sun Niagara consumes 8 W [65] and the Alpha in-order cores EV4 and EV5 consume 1.83 W and 4.43 W respectively at 100 nm technology [50]. Hence, we present most of our results for a range of in-order core power numbers and highlight the conclusions for two in-order core implementations: one that consumes an optimistic 7 W and another that consumes a pessimistic 15 W.

Figure 4.2(a) shows the floorplan of a baseline 2D implementation of a single-die single-core processor model (model 2d-a) and Figure 4.2(b) shows the floorplan of the upper die in a 3D chip that includes the in-order core (upper die of model 3d-2a). The layout in Figure 4.2(c) shows a second baseline 2D layout (2d-2a) that has as much total area and the same structures as the 3D processor implementation, but implemented in 2D. The floorplan of the out-of-order core is modeled loosely after the EV7 core's floorplan. Since the original EV7 floorplan is based on a 130 nm process, we scale the area down using nonideal scaling factors for SRAM and logic, as described in [66] for a 65 nm process. To reduce temperature and the length of the intercore interconnects in 3D, the intercore buffers of the checker core on the upper die are placed as close as possible to the cache structures of the leading core.

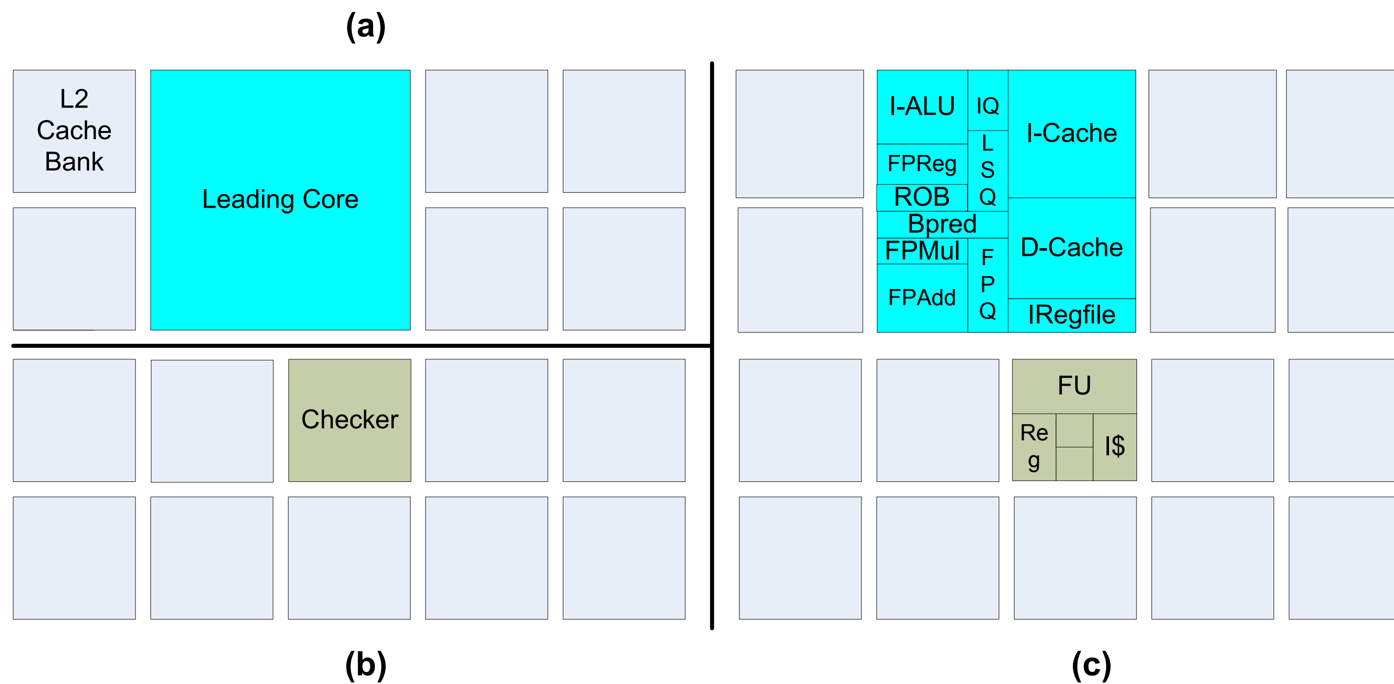


Figure 4.2. Floorplans for each processor model: (a) represents the baseline 2d-a model and the bottom die of the 3d-2a model, (b) represents the upper die of the 3d-2a model, (c) represents the 2d-2a model.

We have also attempted to reduce temperature by placing L2 cache banks above the hottest units in the leading core, when possible.

Figure 4.3 shows the peak temperatures averaged across all benchmark programs for the two baseline 2D organizations and for the 3D implementation (for various checker core power values). Per-benchmark results are shown in Figure 4.4.

The horizontal bar (labeled "2d-a") shows the temperature of the baseline 2D chip that has a 6 MB L2 cache and no checker core (the chip that will be purchased by customers with low reliability requirements). The light bars (labeled "3d-2a") represent the reliable chip with an additional checker core and 9 MB L2 cache snapped on the "2d-a" baseline. The dark bars (labeled "2d-2a") represent a 2D implementation of the 3D chip (with 15 MB L2 cache and checker core). If the checker power is low enough (less than 10 W), the 2d-2a baseline has a lower temperature than the 2d-a baseline because of the lateral heat spreading promoted by the many (relatively) cool L2 cache banks and its larger heat sink. Each L2 cache bank is also cooler in the 2d-2a case as the same number of L2 accesses are distributed across more banks. We also observe that the thermal overheads due to 3D stacking are not high for most checker core values. For a checker core

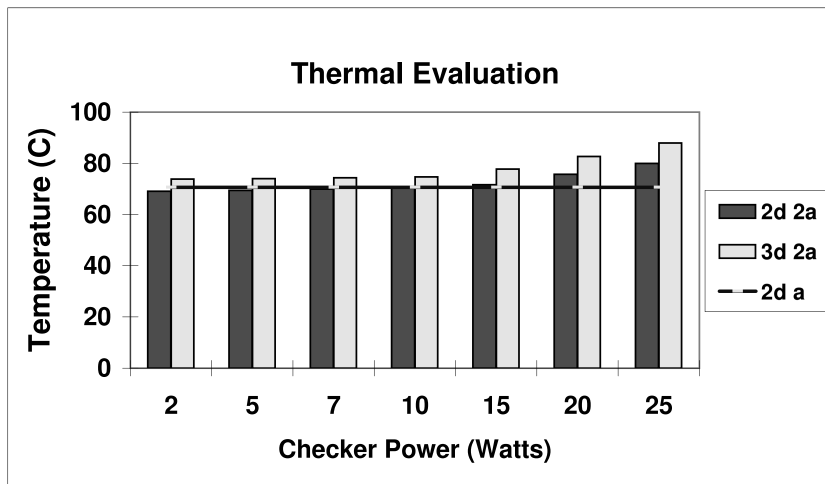


Figure 4.3. Thermal overhead analysis of 3D checker

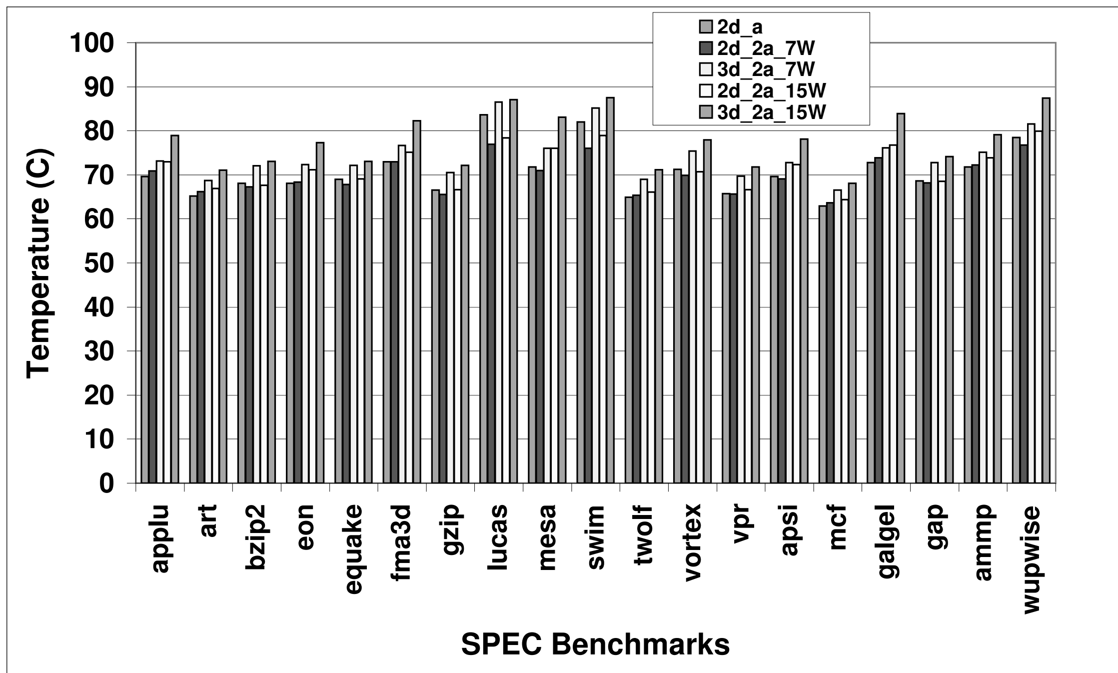


Figure 4.4. Thermal overhead analysis of 3D checker for each benchmark

dissipating 7 W, the 3D model is only 4.5 degrees hotter than the 2d-2a model and 4 degrees hotter than the 2d-a model. If the checker core consumes roughly half the power of the leading core (15 W), it is 7 degrees hotter than the 2d-a model. Thus, 3D integration does have a cost. Even though 3D integration enables a 3.4 W power reduction, relative to the 2d-2a model, because of shorter interconnects (clarified in the next subsection), it still leads to higher power densities. Not surprisingly, the temperature increase is a strong function of the power density of the hottest block in the in-order core. If the power density of the 15 W checker is further doubled, the temperature rise can be as high as 19 degrees, although, this is an admittedly pessimistic scenario. Higher temperatures will either require better cooling capacities or dynamic thermal management (DTM) that can lead to performance loss.

We also modeled the effect of temperature on leakage power in L2 cache banks. Very few banks in 3d have higher temperature if stacked above a hot unit in the

main core. We found the overall impact of temperature on leakage power of caches to be negligible.

It is possible to lower temperatures by not employing the space on the top die for L2 cache. For the 3d-2a case with the 7 W (15 W) checker core, if the top die's checker core is surrounded by inactive silicon, the temperature reduces only by 2 degrees (1 degree). As we show in the next subsection, the impact of this choice on performance is marginal given the working sets of the SPEC2k applications. However, it is unlikely that manufacturers will choose to waste silicon in this manner, especially since the cache needs of multicore chips will likely be higher. Hsu et al. [67] show that for heavily multithreaded workloads, increasing the cache capacity by many mega-bytes yields significantly lower cache miss rates. A second approach to lowering temperature is to move the checker core to the corner of the top die (while incurring a higher cost for intercore communication). Our experiments show that this approach helps reduce temperature by about 1.5 degrees.

A low-power (7 W) checker core causes a temperature increase of only 4.5 degrees (compared to an unreliable baseline), while a high-power (15 W) checker core can increase temperature by 7 degrees.

4.3.3 Performance

Figure 4.5 shows performance of the leading core for all three models without any thermal constraint and using the NUCA policy where sets are distributed across banks. We also show the performance of a model (3d-Checker) where the top die consists of only the checker core and no additional L2 cache banks. As described in Section 4.2, the checker core is operated at a frequency such that it rarely stalls the leading thread and imposes a negligible performance overhead. The 3d-checker model confirms this fact as its performance is the same as the 2d-a baseline that does not have a checker core and has an equal cache capacity. Hence, most of the performance differences in the 2d-a, 2d-2a, and 3d-2a models can be attributed to the L2 cache organization. For the SPEC benchmarks, a 15 MB L2 cache does

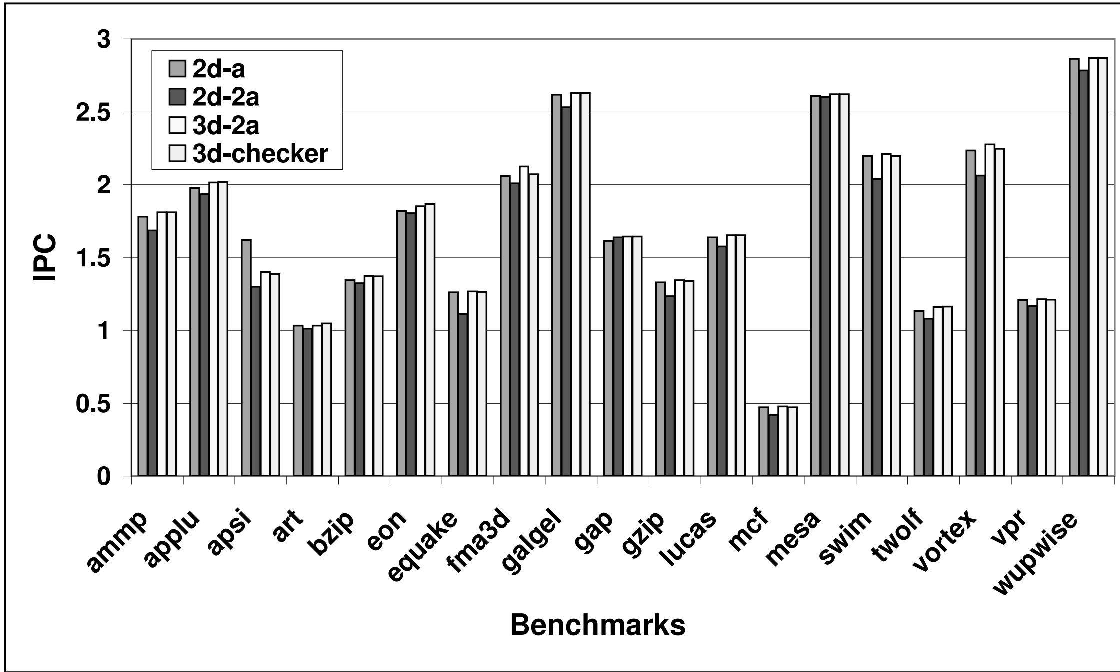


Figure 4.5. Performance evaluation (with a NUCA policy that distributes sets across banks).

not offer much better cache hit rates than a 6 MB cache (the number of L2 misses per 10K instructions reduces from 1.43 to 1.25). The 2d-2a model performs a little worse than the 2d-a model because cache values are more spread out and the average latency for an L2 hit increases from 18 cycles (2d-a) to 22 cycles (2d-2a). The move to 3D does not help reduce the average L2 hit time compared to 2d-a as the average horizontal distance to the banks remains the same. Relative to the 2d-2a baseline, the 3D chip yields a 5.5% performance improvement.

Even with a NUCA policy that distributes ways, we observe the same relative results between 2d-a, 2d-2a, and 3d-2a. However, the performance with the the distributed-way policy is slightly (< 2%) better than with the distributed-set policy because data blocks tend to migrate closer to the L2 cache controller (especially since the program working sets for SPEC2k are typically smaller than the L2 capacity).

The 3D reliable processor causes a temperature increase that can impact packaging/cooling cost, leakage power, and even reliability. Hence, it is also important

to evaluate the various processor models while maintaining a constant thermal constraint. We execute the 3D processor at a lower voltage and frequency than the 2D baseline processor until its average peak temperature matches that of the baseline. Similar to the methodology in [20], we assume that voltage and frequency scale linearly for the voltage ranges considered here. We observed that a 3D processor with a 7 W checker (15 W checker) operating at a frequency of 1.9 GHz (1.8 GHz) has the same thermal characteristics as a 2D baseline processor operating at a frequency of 2 GHz. Thus, assuming constant thermals, the design of a 3D reliable processor entails a performance loss of 4.1% for the 7 W in-order checker core and 8.2% for the 15 W checker core (the performance loss is a little less than the frequency reduction because memory latency is unchanged).

Without a thermal constraint, the second die has a negligible impact on performance because the additional cache space neither improves hit rates nor degrades average access time. The extra cache space may be more valuable if it is shared by multiple threads in a large multicore chip [67]. With a thermal constraint, the performance loss (4%/8%) is a function of the checker core power (7 W/15 W).

4.3.4 Interconnect Evaluation

We next consider the changes to interconnect power and area as a result of the 3D organization. Our methodology is based on the modeling described in [16].

To quantify the requirement for the number of die-to-die (d2d) vias, we compute the data that gets communicated across both dies per cycle. As described in the previous section, the leading core sends load values, register values, and branch outcomes to the checker core and the checker core sends store values back. Table 4.4 quantifies the maximum per-cycle data bandwidth of these interdie transfers and where the d2d vias need to be placed. The maximum bandwidth is a function of the core's issue/execution width. For a 4-wide core, 1025 vias are required between the leading and checker cores. In addition, we introduce a 384-bit via pillar to transmit addresses and data between the L2 cache controller on the lower die and the L2 banks on the upper die (64-bit address, 256-bit data, and 64-bit control signals).

Table 4.4. D2D interconnect bandwidth requirements

Data	Width	Placement of via
Loads	*load_issue_width*$\times$ 64	LSQ
Branch outcome	*branch_pred_ports*$\times$ 1	Bpred
Stores	*store_issue_width*$\times$ 64	LSQ
Register values	*issue_width*$\times$ 192	Register File
Die to die L2 cache transfer	384	L2 Cache Controller

The d2d vias impose a minor power and performance overhead. State-of-the-art 3D integration employs thin dies that results in d2d via lengths from 5 μm to 20 μm [68]. Assuming a d2d via length of 10 μm, the worst-case capacitance of a d2d via surrounded by 8 other vias is computed as 0.594e-15 $F/\mu m$. The capacitance for a via of length 10 μm is therefore 0.59e-14 F. The dynamic power at 65 nm (2 GHz frequency and 1 V) is calculated to be 0.011 mW for each d2d via. The total power consumed by all 1409 vias adds up to only 15.49 mW. State-of-the-art width of each via is 5 μm [68]. The area overhead for all the vias is 0.07 mm^2 assuming that the width and spacing between each via is 5 μm.

We next look at metalization area savings due to reduction in horizontal interconnect length. The total length of horizontal wires in 2D for intercore communication is 7490 mm and in 3D is 4279 mm. If we assume that these are global wires then the total metalization area (given by $pitch \times length$, pitch of 210 nm at 65 nm technology) in 2d is 1.57mm^2 and in 3d is 0.898mm^2 resulting in net metal area savings of 42%. Similarly, we compute net metal savings in L2 interconnect for a NUCA cache. The 2d-a baseline has the least metal area requirement of 2.36mm^2, and the 2d-2a baseline has a requirement of 5.49mm^2. The 3D NUCA layout requires 4.61mm^2, about 16% lower than 2d-2a.

Finally, we look at power consumption of these metal wires. We assume that the interconnect is power-optimized and compute the total power based on methodology in [69]. We observe that the L2 cache and intercore interconnect power consumed by the 2d-a baseline is 5.1 W, 2d-2a consumes 15.5 W, and 3d-2a consumes 12.1 W. The net power savings due to 3D integration compared to 2d-2a is 3.4 W. The

additional 10 W interconnect power in moving from 2d-a to 3d-2a is primarily because of the additional L2 banks – the transfer of register and load values incurs an overhead of only 1.8 W.

The power and area overheads of the interdie vias are marginal. The horizontal interconnects that feed the checker core consume only 1.8 W. The interconnects for the larger cache on the top die are responsible for an additional 9 W of power.

4.3.5 Conservative Timing Margins

In the previous section, we argued that the checker core is capable of detecting a single dynamic timing error that manifests in an incorrect register value. Recovery is also almost always possible with an ECC-protected register file, unless the timing error (or soft error) happens when writing the result into the checker core's register file after verification. Unfortunately, dynamic timing errors are often correlated and multiple such errors in the same or successive cycles can happen. The first such error will be detected by the checker core, but recovery may not be possible if the checker's register file state is corrupted beyond repair (ECC may not be able to repair multiple bit flips, nor a result being written to the wrong register). Further, the checker core is not resilient to faults in the control path. Hence, the processor's ability to recover from an error can be greatly improved if the checker core is made more resilient to dynamic timing errors and soft errors. In this subsection, we explore the option of introducing conservative timing margins in every pipeline stage of the checker core. First, this reduces the chances of a soft error as there is enough slack within the cycle time to allow the circuit to re-stabilize after the pulse and mask the error. Second, even with timing variations, it improves the likelihood that the result is ready when the pipeline latch receives the clock edge.

We first assume that the peak frequency of the checker core is maintained the same as the leading core's peak frequency. To introduce a conservative timing margin in each stage, we must perform less work in each stage. Hence, the work required of the checker core is now distributed across many more pipeline stages, the checker is implemented as a deep pipeline. Note that the deep pipeline is not

being exploited to improve the checker's clock frequency, but to provide more slack in each pipeline stage.

Unfortunately, the power dissipation of a processor increases as its pipeline depth increases due to the higher number of required latches and increased bypassing complexity. To model the power overheads of deep pipelining, we employ the analytical models proposed by Srinivasan et al. [70] to compute power-optimal pipeline depths. More details of this model and assumptions can be found in the original work [70].

Table 4.5 summarizes the impact of pipeline depth on total power, relative to a baseline pipeline that has a cycle time of 18 FO4. While performing less work in each pipeline stage helps reduce error rates, the power cost is enormous. Even if the circuits in each stage take up 14 FO4 gate delays, the power consumption of the checker core increases by 50%. Besides, it may be difficult to pipeline the structures of the processor into the required number of stages. Hence, we don't consider this option any more in this work.

Fortunately, our checker core is designed such that it often operates at a frequency much lower than the peak frequency. As described in Chapter 3, the checker core lowers its frequency if the slack between the leading and checker core decreases. In Figure 4.6, we show a histogram of the percentage of time spent at each frequency level. For most of the time, the checker operates at 0.6 times the peak frequency of 2 GHz. This means that the circuits in each pipeline stage have usually finished their operation within half the cycle time and minor variations in circuit delay will likely not impact the value latched at the end of the pipeline stage. Thus, a natural

Table 4.5. Impact of pipeline scaling on power overheads in terms of baseline dynamic power consumption

Pipeline depth	Dynamic Power Relative	Leakage Power Relative	Total Power Relative
18 FO4	1	0.3	1.3
14 FO4	1.65	0.32	1.97
10 FO4	1.76	0.36	2.12
6 FO4	3.45	0.53	3.98

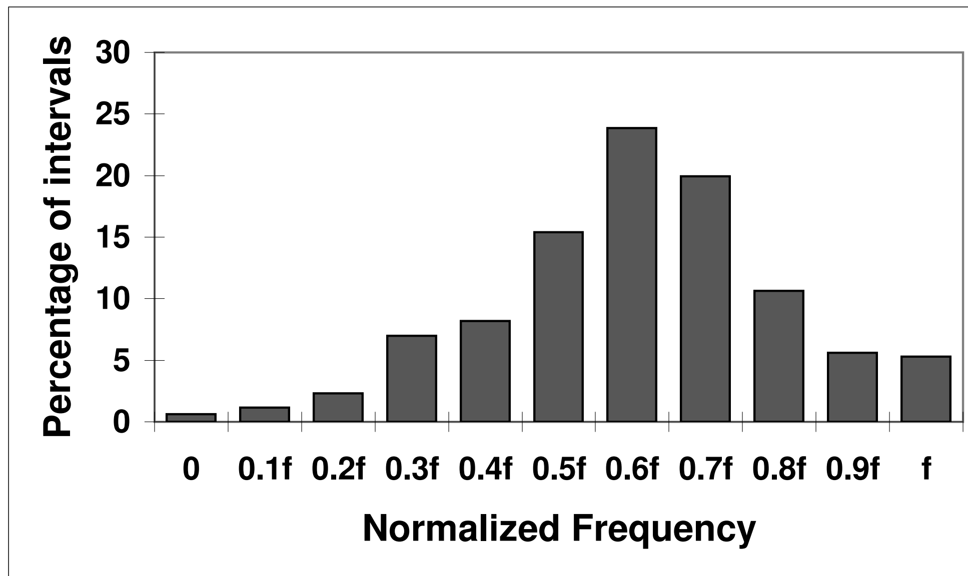


Figure 4.6. Histogram showing percentage of intervals at each normalized frequency.

fall-out of our checker core design is that it is much more resilient to dynamic timing errors and soft errors.

Deep pipelining has an inordinate power overhead. Because of its high ILP, the checker core can typically operate at a fraction of its peak frequency. This allows each pipeline stage to already have plenty of slack and tolerate noise.

4.4 The Impact of Heterogeneity on the Checker

3D integration enables the snap-on functionality of a checker core for customers that require high reliability. It also enables each die to be manufactured in a different process. This has especially favorable implications for the design of a checker core. Unlike the leading core (that must be optimized for performance), the checker core can afford to give up some performance for a manufacturing process that provides high reliability. This allows the checker core to be more resilient to faults in the control path, something that cannot be handled by the register checking process. Further, if the checker core has a much lower probability for an error, its result can be directly employed in case of a disagreement between the leading and checker cores. If the checker core is equally likely to yield an error

as the leading core, recovery requires an ECC-protected checker register file and possibly even a third core to implement triple modular redundancy (TMR). In this section, we consider the use of an older process technology to implement the checker core's die. We have seen that the checker core is already more reliable because it has conservative timing margins – the older process further augments checker core reliability.

The use of an older process technology has the following impact:

- Parameter variations and dynamic timing errors are reduced.
- The critical charge required to switch a transistor's state increases, reducing the probability of a soft error.
- Leakage power is reduced (note that a large fraction of the die is L2 cache that dissipates large amounts of leakage).
- Dynamic power increases because of higher switching capacitance.
- Delay of a pipeline stage increases, possibly necessitating a lower clock speed and poorer performance.

We next analyze each of these factors.

Figure 4.7 shows scaling of per-bit SRAM soft-error rate across process nodes due to neutron and alpha particles. Date presented in [1] shows the probability of per-bit multiple-bit upset (MBU) rate as a function of decreasing critical charge. Even though single-bit error rates per transistor are reducing at future technologies, the overall error rate is increasing because of higher transistor density. The multibit error rate per transistor is increasing at future technologies and can be problematic even with a checker core. Further, Table 4.6 shows ITRS data [3] exhibiting projected variability (as a +/- percentage change from nominal) in threshold voltage, performance, and power, as a function of process technology. All of these data argue for the use of an older technology to implement an error-tolerant checker core die.

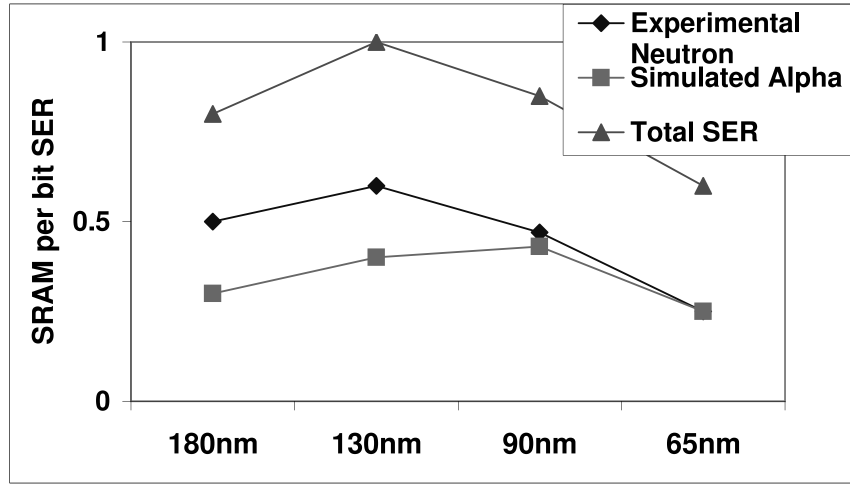


Figure 4.7. SRAM single-bit soft error scaling rate (data from [1])

Table 4.6. Impact of technology scaling on variability

Tech gen nm	Vth Variability	Circuit Performance Variability	Circuit Power Variability
80	26%	41%	55%
65	33%	45%	56%
45	42%	50%	58%
32	58%	57%	59%

To model power consumption of the same core implemented on different process technologies, we employ device model parameters from ITRS reports [3] (summarized in Table 4.7). These were substituted in equations for leakage and dynamic power to generate the relative power estimates for various technologies (listed in Table 4.8).

The data in Table 4.8 show that if the older process is 90 nm, the dynamic power consumption is more than 2x and 3x the dynamic power of the core implemented in a more recent technology (65 nm and 45 nm, respectively). However, leakage power is reduced quite significantly by moving to the older process. Thus, the use of an older technology may increase/decrease the overall power dissipation depending on the relative contribution of leakage and dynamic power. If the die containing the checker core is primarily composed of L2 cache banks, leakage likely accounts for

Table 4.7. Impact of technology scaling on various device characteristics

Tech gen nm	Voltage V	Gate length nm	Capacitance per um	Subthreshold leakage current per um
90	1.2	37	8.79E-16	0.05
65	1.1	25	6.99E-16	0.2
45	1	18	8.28E-16	0.28

Table 4.8. Impact of technology scaling on power consumption

Tech gen nm/nm	Dynamic Power Relative	Leakage Power Relative
90/65	2.21	0.4
90/45	3.14	0.44
65/45	1.41	0.99

a large fraction of die power. It must be noted that these arguments are less clear for newer process technologies as new materials are expected to alleviate leakage concerns (as is observed for the comparison of 65 nm and 45 nm technology nodes in Table 4.8).

For our power-hungry checker core and L2 cache models, we observed a total power dissipation of 18 W for the checker die at 65 nm technology (14.5 W for the checker core and 3.5 W for the 9 MB of L2 cache banks). If we maintain a constant die area and migrate to 90 nm technology, the checker core now consumes 23.7 W and the 5 MB of L2 cache consumes 1.2 W, leading to a net power increase of 6.9 W. In spite of the power increase, we observe a drop in temperature (of 4 ℃, compared to the 3D chip with homogeneous 65 nm technology dies). This is a result of an increase in checker core area and an overall reduction in its power density.

Finally, the older process technology causes an increase in the delay of each circuit and pipeline stage. We have assumed a peak clock frequency of 2 GHz in this study, meaning each pipeline stage can take as long as 500 ps (including overheads for latch, skew, and jitter). If the checker core is now implemented in a 90 nm process instead of a 65 nm process, a pipeline stage that took 500 ps in 65 nm process, now takes 714 ps. This limits the peak frequency of the checker

core to 1.4 GHz. Our simulations have shown that a checker core needs to operate at an average frequency of only 1.26 GHz to keep up with a 2 GHz leading core. If the checker core begins to fall behind the trailer, it must increase its frequency, but is limited to a peak frequency of only 1.4 GHz. This modification causes only a minor slowdown to the leading core (3%). The longer delays for the circuits imply less slack until the next clock edge. On average, given the frequency profile of the checker core, there still remains nontrivial slack in each pipeline stage even with the slower circuits.

The L2 cache banks in the older process operate at the same frequency (2 Ghz) as the other banks on the lower die. The access time for each L2 cache bank in the older process increases by a single cycle. Overall cache access times are not greatly impacted as the fewer cache banks lead to fewer hops on average.

An older process increases overall power consumption, but reduces the power density for the hottest units on the chip, allowing overall temperature to decrease by up to 4 degrees. Thus, with a constant thermal constraint, overall performance actually improves over the design with homogeneous dies. A 15 W checker core implemented in an older process will either increase temperature by 3 degrees or suffer a performance loss of 4% under a constant thermal constraint (relative to the 2D baseline). The older process is more resilient to faults and a large portion of the execution continues to enjoy conservative timing margins for the pipeline stages.

We observed that the DFS heuristic for matching throughput can play a significant role in power and thermal properties of the checker. An aggressive heuristic that can slow down the checker core further results in lower temperature values. However, such an aggressive mechanism can stall the main core more frequently and result in performance loss compared to an unreliable 2D baseline. In our models, we have employed a less aggressive heuristic that does not degrade the main core's performance by itself. This heuristic does lead to higher temperatures, which in turn lead to thermal emergencies and lower performance.

4.5 Related Work

Many fault-tolerant architectures [34–36, 38, 40, 54–56, 71] have been proposed over the last few years. Not all of these designs are amenable to a convenient 3D implementation. Most reliable implementations that execute the redundant thread on a separate core will likely be a good fit for a 3D implementation. While our evaluation focuses on the specific implementation outlined in [71], the results will likely apply to other RMT implementations that provide low-overhead redundancy on a separate core.

There has been much recent work on architectural evaluations involving 3D chips. Many have focused on improving single-core performance and power [12, 20–22]. Some of this attention has focused on implementing a cache in 3D [72–74], even in the context of a multicore NUCA layout [30]. Few studies have considered using additional dies entirely for SRAM or DRAM [19, 20, 32].

The work in this study is most closely related to the work by Mysore et al. [16]. That work focuses on implementing software profiling and analysis engines on the second die. Further, it does not explore microarchitectural design choices for the second die and does not consider heterogeneous dies. Many of the findings of this study are specific to the implementation of a checker core on the second die: most notably, the effect of technology, pipelining, and frequency scaling on the ability of the checker core to resist transient and dynamic timing errors. A software profiling and analysis application, on the other hand, is different in its needs: it may be more tolerant of errors and less tolerant of performance slowdowns.

4.6 Summary

In this chapter, we propose a 3D reliable processor organization and evaluate many of its design considerations. A checker core on the upper die helps reduce interconnect lengths and part of the upper die can also be employed to increase L2 cache size. The isolation of the checker core to a separate die reduces the impact on the leading core's layout, wiring, and cycle time. The snap-on functionality allows customers the option to detect and recover from failures, while not impacting the

manufacturing cost for customers that are less concerned about reliability. The most significant impact of this design is a temperature increase of up to 7 ℃ or alternatively, a performance loss of up to 8% if a constant thermal constraint is imposed. The impact on temperature and performance is much less if we assume a simple low-power microarchitecture for the in-order core. We also show that the checker core is more error-resilient because it typically operates at a much lower frequency. Error tolerance can be further increased by implementing the upper die on an older process that suffers less from dynamic timing errors and soft errors. The move to an older technology increases power consumption, but reduces temperature because the power density of the hottest block is lowered. This helps limit the overhead of the checker core to a 3 ℃ temperature increase or a 4% performance loss. Our results largely argue in favor of heterogeneous dies for this specific application domain of low-overhead redundancy.

CHAPTER 5

A 3D STACKED RECONFIGURABLE CACHE HIERARCHY DESIGN

In this chapter, we focus on the second major component of this thesis where we demonstrate how 3D die stacking can be leveraged to design a balanced cache hierarchy that can optimize capacity and communication in large caches. Recent studies that proposed stacking of DRAM die over SRAM die focused mainly on reaping the bandwidth and fast interconnect advantage enabled by 3D technology. We investigate this mixed technology stacking from a different angle and propose the design of a heterogeneous SRAM-DRAM cache. This hybrid cache combines the benefit of DRAM's density (8x times SRAM) to get the capacity and SRAM's superior power/delay properties. The central idea in this proposal is to enable the DRAM bank stacked above the SRAM bank to grow the last level cache's size vertically in the third dimension. Such an implementation is not possible with a traditional 2D chip. Such a cache design allows dynamic reconfiguration of the hybrid cache based on the application's demand for cache capacity. Apart from highlighting this novel application of mixed process 3D integration, this proposal also combines interesting techniques such as OS-based page coloring and tree on-chip interconnect network to optimize the horizontal communication between cache banks.

The contents of this chapter are organized as follows. Section 5.2 provides basic background on recent innovations in multicore cache design and related work. Section 5.3 describes our proposed cache architecture. Results are discussed in section 5.4 and we summarize this work in Section 5.5.

5.1 Leveraging 3D Technology for Cache Hierarchy Design

The design of cache hierarchies for multicore chips has received much attention in recent years (for example, [75–80]). As process technologies continue to shrink, a single chip will accommodate many mega-bytes of cache storage and numerous processing cores. It is well known that the interconnects that exchange data between caches and cores represent a major bottleneck with regard to both power and performance. Modern Intel chips already accommodate up to 27 MB of cache space [43]; interconnects have been attributed as much as 50% of total chip dynamic power [8]; on-chip networks for large tiled chips have been shown to consume as much as 36% of total chip power [81, 82]; long on-chip wires and router pipeline delays can lead to cache access times of many tens of cycles [3, 83]. Not only will we require intelligent mechanisms to allocate cache space among cores, we will also have to optimize the interconnect that exchanges data between caches and cores. This work makes an attempt at addressing both of these issues.

3D stacking of dies has been demonstrated as a feasible technology [12] and is already being commercially employed in some embedded domains [13, 14]. In most commercial examples, 3D is employed to stack DRAM memory upon CPU cores [13, 14]. This is especially compelling because future multicores will make higher memory bandwidth demands and the interdie interconnect in a 3D chip can support large data bandwidths. Early projections for Intel's Polaris 80-core prototype allude to the use of such 3D stacking of DRAM to feed data to the 80 cores [84]. Given the commercial viability of this technology, a few research groups have already begun to explore the architectural ramifications of being able to stack storage upon CPUs [19, 30–33].

In this chapter, we first postulate a physical processor design that is consistent with the above trends. We then take advantage of the following three key innovations to architect a cache hierarchy that greatly reduces latency and power: (i) we employ page coloring to localize data and computation, (ii) we propose the use of cache reconfiguration to accommodate large working sets for some cores, (iii) we

identify a network topology that best matches the needs of data traffic and incurs low delay and power overheads.

The proposed processor employs three dies stacked upon each other (see Figure 5.1). The lowest die contains the processing cores (along with the corresponding L1 caches). The second die is composed entirely of SRAM cache banks (forming a large shared L2 cache) and employs an on-chip network so that requests from the CPU can be routed to the correct bank. The third die is composed of DRAM banks that serve to augment the L2 cache space provided by the second SRAM die. It is also possible to stack many more DRAM dies upon these three dies to implement main memory [33], but we regard this as an orthogonal design choice and do not consider it further in this study.

The L2 cache banks are organized as a nonuniform cache architecture (NUCA) [85]. The request from the processing core is transmitted to the SRAM die through interdie vias. From here, the request is propagated to the appropriate bank through the on-chip network and the latency for the access is a function of the proximity of this bank to the processing core. Many recent papers have explored various mechanisms to reduce average access times in a NUCA cache [75, 77, 78, 80, 86, 87]. Most dynamic (D-NUCA) mechanisms can cause data to be placed anywhere on the chip, requiring search mechanisms to locate the data. We disregard these algorithms because of the complexity/cost of search mechanisms and resort to a static-NUCA (S-NUCA) organization, where a given physical address maps to a unique bank in the cache (the physical address maps to a unique bank and set within that bank; the ways of the set may be distributed over multiple subarrays within that bank). To improve the proximity of storage and computation, we employ page coloring to ensure that data are placed in "optimal" banks. The idea of employing page coloring to dictate data placement in a NUCA cache was proposed in a recent paper by Cho and Jin [88]. Ideally, a low-overhead run-time mechanism would be required to estimate the usage of a page so that pages can be dynamically migrated to their optimal locations [89]. The design of such mechanisms is a nontrivial problem in itself and beyond the scope of this work. For this work, we carry out an off-line

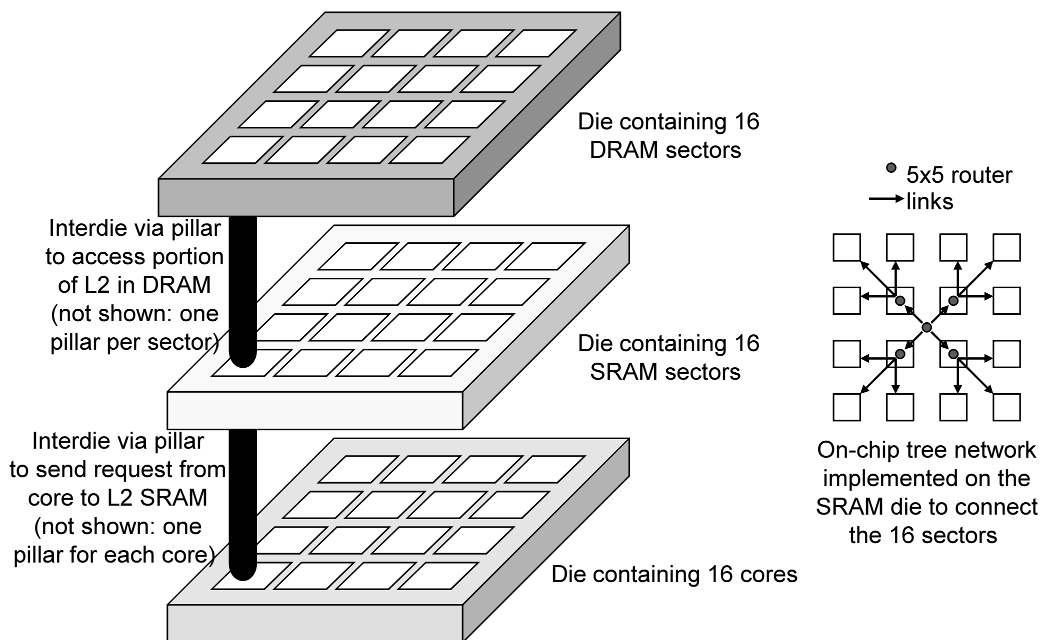


Figure 5.1. Stacked 3D processor layout. The bottom die contains the 16 cores, the second die contains 16 SRAM banks, and the top die contains 16 DRAM banks. Interdie via pillars (one for each bank) are used to implement vertical connections between cores and DRAM/SRAM banks. Horizontal communication happens on the on-chip tree network (shown on the right) implemented on the second die. There are no horizontal connections between cores (banks) on the bottom (top) die.

analysis to identify pages that are private to each core and that are shared by multiple cores. Private pages are placed in the bank directly above the core and shared pages are placed in one of four central banks.

With the above page coloring mechanism in place, we expect high data locality and most cores will end up finding their data in the L2 cache bank directly above. In a multiprogrammed workload, each core may place different demands on the L2 cache bank directly above. In a multithreaded workload, the centrally located cache banks will experience higher pressure because they must accommodate shared data in addition to the data that is private to the corresponding cores. These varied demands on each cache bank can perhaps be handled by allowing a core to spill some of its pages into the adjacent banks. However, this not only increases the average latency to access that page, it also places a higher bandwidth demand on the interbank network, a trait we are striving to avoid (more on this in the next

paragraph). Hence, we instead spill additional pages into the third dimension – to the DRAM bank directly above the SRAM cache bank. Note that the DRAM bank and SRAM bank form a single large vertical slice in the same L2 cache level. When the DRAM space is not employed, each SRAM cache bank accommodates 1 MB of cache space. If this space is exceeded, the DRAM bank directly above is activated. Since DRAM density is eight times SRAM density, this allows the cache space to increase to roughly 9 MB. While DRAM has poorer latency and power characteristics than SRAM, its higher density allows a dramatic increase in cache space without the need for many more stacked dies (that in turn can worsen temperature characteristics). Further, we architect the combined SRAM-DRAM cache space in a manner that allows nonuniform latencies and attempts to service more requests from the faster SRAM die. DRAM bank access itself has much lower cost than traditional DRAM main memory access because the DRAM die is partitioned into many small banks and a single small DRAM bank is looked up at a time without traversing long wires on the DRAM die. This results in a heterogeneous reconfigurable cache space, an artifact made possible by 3D die stacking. A reasonable alternative would be the implementation of a three-level cache hierarchy with the top DRAM die serving as an independent L3 cache. A static design choice like that would possibly complicate cache coherence implementations, incur the latency to go through three levels for many accesses, and reduce the effective capacity of the L2 because of the need for L3 tags. We believe that the overall design is made simpler and faster by growing the size of the L2 cache bank on a need basis.

Finally, we examine the traffic patterns generated by the cache hierarchy described above. Most requests are serviced by the local SRAM and DRAM cache banks and do not require long traversals on horizontal wires. Requests to shared pages are directed towards the centrally located banks. Requests are rarely sent to nonlocal noncentral banks. Such a traffic pattern is an excellent fit for a tree network (illustrated in Figure 5.1). A tree network employs much fewer routers and links than the grid network typically employed in such settings. Routers and links

are cumbersome structures and are known to consume large amounts of power and area [81, 82] – hence, a reduction in routers and links has many favorable implications. Tree networks perform very poorly with random traffic patterns, but the use of intelligent page coloring ensures that the traffic pattern is not random and best fits the network topology. A tree network will likely work very poorly for previously proposed D-NUCA mechanisms that can place data in one of many possible banks and that require search mechanisms. A tree network will also work poorly if highly pressured cache banks spill data into neighboring banks, making such a topology especially apt for the proposed design that spills data into the third dimension.

The contributions of the work in this chapter are:

- A synergistic combination of page coloring, cache reconfiguration, and on-chip network design that improves performance by up to 62%.
- The design of a heterogeneous reconfigurable cache and policies to switch between configurations.

5.2 Background and Related Work

Most future large caches are expected to have NUCA architectures [85]. A large shared L2 or L3 cache can either be placed in a contiguous region or split into slices and associated with each core (tile). Early designs split the ways of a set across multiple banks, allowing a given block to have multiple possible residences. Policies were proposed to allow a block to gravitate towards a way/bank that minimized access time (D-NUCA [85]). However, this led to a nontrivial search problem: a request had to look in multiple banks to eventually locate data. A static-NUCA (S-NUCA) design instead places all ways of a set in a single bank and distributes sets across banks. Given a block address, the request is sent to a unique bank, that may or may not be in proximity. In a recent paper, Cho and Jin [88] show that intelligent page coloring can influence address index bits so that the block is mapped to a set and bank that optimizes access time. The work in this research is

built upon the page coloring concept to improve access times and eliminate search. Other papers that attempt to improve data placement with a D-NUCA approach include [75, 77, 80, 86, 87]. A number of papers also attempt to improve multicore cache organizations by managing data co-operatively within a collection of private caches [76, 79, 90]. In these papers, if a core's private cache cannot provide the necessary capacity, blocks are spilled into the private caches of neighboring cores.

Recent papers have also proposed innovative networks for large caches. Jin et al. [91] propose a halo network that best meets the needs of a single-core D-NUCA cache, where requests begin at a cache controller and radiate away as they perform the search. Beckmann and Wood [92] propose the use of transmission lines to support low-latency access to distant cache banks. Muralimanohar and Balasubramonian [4] propose a hybrid network with different wiring and topologies for the address and data networks to improve access times. Guz et al. [93] propose the Nahalal organization to better manage shared and private data between cores.

A number of recent papers have proposed cache hierarchy organizations in 3D. Li et al. [30] describe the network structures required for the efficient layout of a collection of cores and NUCA cache banks in 3D. Some bodies of work implement entire SRAM cache structures on separate dies [19, 31, 32]. Loh [33] proposes the changes required to the memory controller and DRAM architecture if several DRAM dies are stacked upon the CPU die to implement main memory. Ours is the first body of work that proposes reconfiguration across dies and combines heterogeneous technologies within a single level of cache.

Prior work on reconfigurable caches has been restricted to a single 2D die and to relatively small caches [94–96]. Some prior work [97–99] logically splits large cache capacity across cores at run-time and can be viewed as a form of reconfiguration.

5.3 Proposed Ideas

5.3.1 Proposed NUCA Organization

We first describe the basic model for access to the L2 cache in our proposed implementation. As shown in Figure 5.1, the bottom die contains 16 cores. The

proposed ideas, including the tree topology for the on-chip network, will apply to larger systems as well. We assume 3.5 mm^2 area for each core at 32 nm technology, based on a scaled version of Sun's Rock core [100]. Each die is assumed to have an area of around 60 mm^2. Each core has its own private L1 data and instruction caches. An L1 miss causes a request to be sent to the L2 cache implemented on the dies above. The SRAM die placed directly upon the processing die is partitioned into 16 banks (we will later describe the role of the DRAM die). Based on estimates from CACTI 6.0 [2], a 1 MB SRAM cache bank and its associated router/controller have an area roughly equal to the area of one core. Each bank may itself be partitioned into multiple subarrays (as estimated by CACTI 6.0) to reduce latency and power. Each bank is associated with a small cache controller unit and a routing unit. On an L1 miss, the core sends the request to the cache controller unit directly above through an interdie via pillar. Studies [18, 33] have shown that high bandwidth vias can be implemented and these vias have pitch values as low as 4 μm [18].

The L2 cache is organized as a static-NUCA. Four bits of the physical address are used to map a data block to one of the 16 banks. As a result, no search mechanism is required – the physical address directly specifies the bank that the request must be routed to. Once an L2 cache controller receives a request from the core directly below, it examines these four bits and places the request on the on-chip network if destined for a remote bank. Once the request arrives at the destination bank, the cache subarrays in that bank are accessed (more details shortly) and data is returned to the requesting core by following the same path in the opposite direction. The L2 tag maintains a directory to ensure coherence among L1 caches and this coherence-related traffic is also sent on the on-chip network.

It must be noted that the baseline model described so far is very similar to tiled multicore architectures that are commonly assumed in many papers (for example, [80]). These are typically referred to as logically shared, but physically distributed L2 caches, where a slice of L2 cache is included in each tile. The key difference in our model is that this slice of L2 is separated into a second SRAM die.

5.3.2 Page Coloring

An S-NUCA organization by itself does not guarantee low access times for L2 requests. Low access times can be obtained by ensuring that the data requested by a core is often placed in the cache bank directly above. Page coloring is a well-established OS technique that exercises greater control on the values assigned to bits of the physical address. Traditionally, page coloring has been employed to eliminate the aliasing problem in large virtually indexed caches. In the context of an S-NUCA cache, page coloring can be employed to influence the four bits of the physical address that determine the bank within the NUCA cache. If the entire L2 cache size is 16 MB (each bank is 1 MB) and is four-way set-associative with 64 byte line size, a 64-bit physical address has the following components: 6 bits of offset, 16 bits of set index, and 42 bits of tag. If the page size is 4 KB, the 12 least significant bits are the page offset and the 52 most significant bits are the physical page number. The four most significant bits of the set index are also part of the physical page number. These four bits can be used to determine the bank number. Since the OS has control over the physical page number, it also has control over the bank number.

For pages that are accessed by only a single core, it is straightforward to color that page such that it maps to the bank directly above the core. For pages that are accessed by multiple cores, the page must ideally be placed in a bank that represents the center-of-gravity of all requests for that page. Creating such a mapping may require page migration mechanisms and hardware counters to monitor a page's access pattern and cache bank pressure. These are nontrivial policies that are beyond the scope of this work. For now, we assume that mechanisms can be developed to closely match the page allocation as computed by an off-line oracle analysis. In other words, the optimal cache bank for a page is precomputed based on the center-of-gravity of requests for that page from various cores. As a result, shared data in multithreaded workloads tend to be placed in the four banks in the center of the chip. We also devise policies to map a shared instruction (code) page either in the central shared bank or as replicated read-only pages in each

bank. We evaluate the following three (oracle) page coloring schemes (also shown in Figure 5.2). Note that in all these schemes, private pages are always placed in the bank directly above; the differences are in how shared data and instruction pages are handled.

- **Share4:D+I :** In this scheme, we employ a policy that assigns both shared data and instruction pages to the central four banks. The shared bank is selected based on the proximity to the core that has maximum accesses to that page. If the program has a high degree of sharing, then we expect the central four banks to have increased pressure. This may cause the central banks to enable their additional DRAM cache space.

- **Rp:I + Share4:D :** In this scheme, we color all shared data pages so they are mapped to central banks. We replicate shared instruction pages and assign them to each accessing core. This causes the bank pressure to increase slightly for all private banks as the working set size of instruction pages is typically very small. This improves performance greatly as code pages are frequently accessed in the last-level cache (LLC) in commercial workloads.

- **Share16:D+I :** In order to uniformly utilize the available cache capacity, we color all shared pages (data and code) and distribute them evenly across all

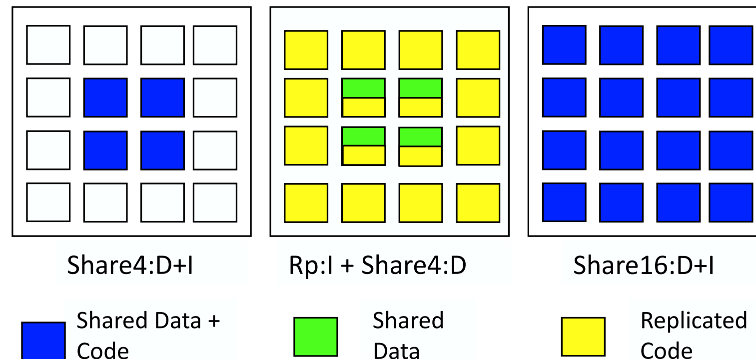


Figure 5.2. Page coloring schemes

16 banks. This page coloring scheme does not optimize for latency for shared pages, but attempts to maximize SRAM bank hit rates.

Note that the above schemes may require minimal OS and hardware support. These schemes rely upon detection of shared versus private pages and code versus data pages.

5.3.3 Reconfigurable SRAM/DRAM Cache

As described, the L2 cache is composed of 16 banks and organized as an S-NUCA. Each bank in the SRAM die has a capacity of 1 MB. If the threads in a core have large working sets, a capacity of 1 MB may result in a large miss rate. Further, centrally located banks must not only service the local needs of their corresponding cores, they must also accommodate shared pages. Therefore, the pressures on each individual bank may be very different. One possibility is to monitor bank miss rates and recolor some pages so they spill into neighboring banks and release the pressure in "popular" banks. Instead, we propose a reconfigurable cache that takes advantage of 3D stacking to seamlessly grow the size of a bank in the vertical dimension. This allows the size of each bank to grow independently without impacting the capacity or access time of neighboring banks. Many proposals of 2D reconfigurable caches already exist in the literature: they allow low access times for small cache sizes but provide the flexibility to incorporate larger capacities at longer access times. The use of 3D and NUCA makes the design of a reconfigurable cache especially attractive: (i) the spare capacity on the third die does not intrude with the layout of the second die, nor does it steal capacity from other neighboring caches (as is commonly done in 2D reconfigurable caches [94, 96]), (ii) since the cache is already partitioned into NUCA banks, the introduction of additional banks and delays does not greatly complicate the control logic, (iii) the use of a third dimension allows access time to grow less than linearly with capacity (another disadvantage of a 2D reconfigurable cache).

While growing the size of a bank, the third die can implement SRAM subarrays identical to the second die, thus allowing the bank size to grow by a factor of

two. But reconfiguration over homogeneous banks does not make much sense: the added capacity comes at a trivial latency cost, so there is little motivation to employ the smaller cache configuration. Reconfiguration does make sense when employing heterogeneous technologies. In this case, we advocate the use of a third die that implements DRAM banks, thus allowing the bank size to grow by up to a factor of nine.[1] By stacking a single DRAM die (instead of eight SRAM dies), we provide high capacity while limiting the growth in temperature and cost, and improving yield. Since DRAM accesses are less efficient than SRAM accesses, dynamic reconfiguration allows us to avoid DRAM accesses when dealing with small working sets.

5.3.3.1 Organizing Tags and Data

The SRAM bank is organized into three memory arrays: a tag array, a data array, and an adaptive array that can act as both tag and data arrays. As with most large L2s, the tags are first looked up and after the appropriate way is identified, the data subarray is accessed. Assuming that each block is 64 bytes and has a 32 bit tag (including directory state and assuming a 36-bit physical address space), the corresponding tag storage is 64 KB. The use of DRAM enables the total bank data storage to grow to 9 MB, requiring a total 576 KB of tag storage. Since tags are accessed before data and since SRAM accesses are cheaper than DRAM accesses, it is beneficial to implement the tags in SRAM. As a result, 512 KB of SRAM storage that previously served as data subarray must now serve as tag subarray. Compared to a typical data array, a tag array has additional logic to perform tag comparison. The data subarray has more H-tree bandwidth for data transfer compared to its tag counterpart. To keep the reconfiguration simple, the proposed adaptive array encompasses the functionalities of both tag and data array.

[1] Published reports claim a factor of eight difference in the densities of SRAM and DRAM [101].

5.3.3.2 Growing Associativity, Sets, Block-Size

The increased capacity provided by DRAM can manifest in three forms (and combinations thereof): (i) increased associativity, (ii) increased number of sets, and (iii) increased block size.

The first form of reconfiguration allows the bank to go from four-way to thirty-four-way set associative. Thirty-two data ways are implemented on the DRAM die and two of the original four data ways remain on the SRAM die after half the data subarrays are converted to tag subarrays. Such an approach has the following advantages: (i) dirty lines in the two ways in the SRAM die need not be flushed upon every reconfiguration, (ii) every set in the bank can have two of its ways in relatively close proximity in SRAM. The primary disadvantage is the power and complexity overhead of implementing thirty-four-way set-associativity. We can optimize the cache lookup further by moving the MRU block into the two SRAM ways on every access in the hope that this will reduce average access times. With this policy, a hit in DRAM space will require an SRAM and DRAM read and write. In our evaluations, we employ reconfiguration of the number of ways, but do not attempt the optimization where MRU blocks are moved into SRAM.

The second form of reconfiguration causes an increase in the number of sets from 2K to 16K (we will restrict ourselves to power-of-two number of sets, possibly leading to extra white space on the top die). When cache size is increased, nearly every dirty cache line will have to be flushed. When cache size is decreased, lines residing in the SRAM data subarrays need not be flushed. The large cache organization has a more favorable access time/power for a fraction of the sets that map to SRAM data subarrays. The page coloring mechanism could attempt to color critical pages so they reside in the sets that map to SRAM.

The third form of reconfiguration increases the block size from 64 bytes to 512 bytes (again, possibly resulting in white space). Note that this approach does not increase the tag space requirement, so 1 MB of data can be placed in SRAM, while 7 MB is placed in DRAM. This has the obvious disadvantage of placing higher pressure on the bus to main memory and also higher energy consumption

for accesses. While reconfiguring the number of sets or block size, care must be taken to not change the address bits used to determine the bank number for an address.

Thus, there are multiple mechanisms to reconfigure the cache. The differences are expected to be minor unless the application is highly sensitive to capacity (8 MB versus 8.5 MB) and memory bandwidth. While some mechanisms can escape flushing the entire cache, these savings are relatively minor if cache reconfiguration is performed infrequently. In our evaluations, we focus only on the first reconfiguration approach that changes the number of ways.

It is worth noting that banks in the DRAM die are laid out very similar to banks in the SRAM die. Our estimates for DRAM delay, power, and area are based on CACTI-6.0 and discussions with industrial teams. Table 5.1 summarizes the delay, power, and area of the considered organizations. The DRAM banks can also be statically employed as a level-3 cache. However, this would significantly reduce the size of the SRAM L2 cache as 0.5 MB space on the second die would have to be designated as L3 tags. This may have a negative impact for several applications with moderate working-set sizes (not evaluated in this paper). It will also increase latencies for L3 and memory accesses because tags in multiple levels have to be sequentially navigated.

Having described the specific implementation, the following additional advantages over prior 2D designs [94, 96] are made clear: (i) a dramatic 8x increase in capacity is possible at a minor delay overhead, (ii) only two configurations

Table 5.1. Access time, energy, and area for various cache configurations at a 4 GHz clock derived from [2].

Configuration	Access time (ns)	Total Energy per access(nJ	Area (mm^2)
Baseline SRAM 1 MB bank	3.13	0.699	2.07
Reconfigurable cache (ways) DRAM	6.71	1.4	3.23
Reconfigurable cache (sets) DRAM	6.52	1.36	2.76
Reconfigurable cache (block size) DRAM	5.43	51.19	1.44

are possible, enabling a simpler reconfiguration policy, (iii) each cache bank can reconfigure independently, thus avoiding a flush of the entire L2 all at one time.

5.3.3.3 Reconfiguration Policies

We next examine the design of a reconfiguration policy. The frequency of reconfiguration is a function of the overheads of a cache flush and cache warm-up. Up to 16K cache lines will have to be flushed or brought in upon every reconfiguration. While the fetch of new lines can be overlapped with execution, cache flush will likely have to stall all requests to that bank. A state-of-the-art memory system can handle a peak throughput of 10 GB/s [3]. A complete flush will require a stall of roughly 100 K cycles. For this overhead to be minor, a reconfiguration is considered once every 10 M cycles. Reconfiguration policies are well-studied (for example, [102–105]). We design two simple policies that are heavily influenced by this prior work.

Every 10 M cycles, we examine the following two metrics for each cache bank: bank miss rate and usage. A high bank miss rate indicates the need to grow cache size, while low usage indicates the need to shrink cache size. It is important to pick a low enough threshold for usage, else the configurations can oscillate. Usage can be determined by maintaining a bit per cache block that is set upon access and reset at the start of every 10 M cycle interval (this is assumed in our simulations). There also exist other complexity-effective mechanisms to estimate usage [98].

In an alternative policy, various statistics can be maintained per bank to determine if the application has moved to a new phase (a part of application execution with different behavior and characteristics). If a substantial change in these statistics is detected over successive large time intervals (*epochs*), a phase chase is signaled. Upon a phase change, we simply implement each of the two configuration choices and measure instruction throughput to determine which configuration is better (referred to as *exploration*). Each exploration step has to be long enough to amortize cache warm-up effects. The optimal organization is employed until the next phase change is signaled. If phase changes are frequent, the epoch length is

doubled in an attempt to capture application behavior at a coarser granularity and minimize the overheads of exploration.

Since these policies are strongly based on prior work (most notably [103]), and have been shown to be effective in other domains, we donot focus our efforts on evaluating the relative merits of each of these policies. Because of the large sizes of the caches and epochs, extremely long simulations will be required to observe any interesting artifacts with regard to program phases. Our simulations model the first reconfiguration policy that chooses to grow or shrink cache size based on miss rate and usage, respectively. In practice, we believe that the second reconfiguration policy may be more effective because it avoids the overheads of having to keep track of cache line usage.

5.3.4 Interconnect Design

Most papers on NUCA designs or tiled multicores have employed grid topologies for the interbank network. Grid topologies provide high performance under heavy load and random traffic patterns. This was indeed the case for prior D-NUCA proposals where data could be placed in one of many possible banks and complex search mechanisms were required to locate data. However, with the 3D reconfigurable cache hierarchy and the use of S-NUCA combined with page coloring, the traffic patterns are typically very predictable. For multiprogrammed workloads, most requests will be serviced without long horizontal transfers and for multithreaded workloads, a number of requests will also be serviced by the four central banks. There will be almost no requests made to nonlocal and noncentral banks. With such a traffic pattern, a grid topology is clearly overkill. Many studies have shown that on-chip routers are bulky units. They not only consume a nontrivial amount of area and power [81], commercial implementations also incur delay overheads of four [81] to eight [106] cycles. Hence, it is necessary that we find a minimal topology that can support the required traffic demands. Given the nature of the traffic pattern, where most horizontal transfers radiate in/out of the central banks, we propose the use of a tree topology, as shown in Figure 5.1. This allows the use

of four 5x5 routers in the central banks and one 4x4 router as the root. In addition, the 12 leaf banks need buffers for incoming flits and some control logic to handle flow control for the outgoing flits. Note that this network is only employed on the second die – there are no horizontal links between banks on the first and third dies.

5.4 Results

5.4.1 Methodology

We use a trace-driven platform simulator ManySim [107] for our performance simulations. ManySim simulates the platform resources with high accuracy, but abstracts the core to optimize for speed. The core is represented by a sequence of compute events (collected from a cycle-accurate core simulator) separated by memory accesses that are injected into the platform model. ManySim contains a detailed cache hierarchy model, a detailed coherence protocol implementation, an on-die interconnect model and a memory model that simulates the maximum sustainable bandwidth specified in the configuration. All our simulation parameters are shown in Table 5.2.

CACTI-6.0 [2] is employed for estimating cache area, access latency, and power. We assume a 32nm process for our work. We derive network router power and area overheads from Orion [61]. Each router pipeline is assumed to three stages and link latency is estimated to be three cycles for the grid and tree topologies. We also incorporate a detailed network model into the ManySim infrastructure.

Table 5.2. Simulation parameters

16 Private MLC Cache banks	Each 128KB 4-way 5-cyc
16 LLC SRAM NUCA Cache Banks	1MB 4-way, 13-cyc
SRAM Active power	0.3W
Page Size	4KB
16 DRAM sector	Each 8MB 32-way 30-cyc
DRAM Active Power	0.42W
Core/Bank Area	3.5mm^2
Chip Footprint	56mm^2
Process node	32nm
Frequency	4 GHz

As an evaluation workload, we chose four key multithreaded commercial server workloads: TPC-C, TPC-E, SPECjbb, and SAP. The bus traces for these workloads were collected on a Xeon MP platform where 8 threads were running simultaneously with the last level cache disabled. To simulate our 16-core system, we duplicate the 8-thread workload to run on 16 cores. This results in true application sharing only between each set of 8 cores. We do offset the address space of each 8-thread trace such that there is no address duplication. Thus, our network latencies for shared pages in the baseline are lower as we do not access the most distant bank, and we will likely see better improvements for a true 16-thread workload. We run these workloads for approximately 145 million memory references. Since we assume an oracle page-coloring mechanism, we annotate our traces off-line with the page color and append the page color bits to the address.

5.4.2 Baseline Organizations

We consider three different baseline organizations with varying number of dies:
Base-No-PC: A chip with two dies: the bottom die has 16 cores and the second die has 16 1 MB L2 SRAM cache banks organized as S-NUCA but with no explicit page coloring policy. All the banks have a roughly equal probability of servicing a core's request. A grid topology is assumed for the interbank network.
Base-2x-No-PC: A chip with three dies: the bottom die has 16 cores and the second and third dies contain SRAM L2 cache banks organized as S-NUCA (no explicit page coloring policy). This organization simply offers twice the cache capacity as the previous baseline at the expense of an additional die.
Base-3-level: The DRAM die is used as an L3 UCA cache. The tags for the L3 cache are implemented on the SRAM die, forcing the SRAM L2 cache size to shrink to 0.5 MB. No page coloring is employed for the L2.

5.4.3 Workload Characterization

We first characterize the server workloads to understand their implications on our proposed techniques. Figure 5.3 shows the percentage of shared pages in all

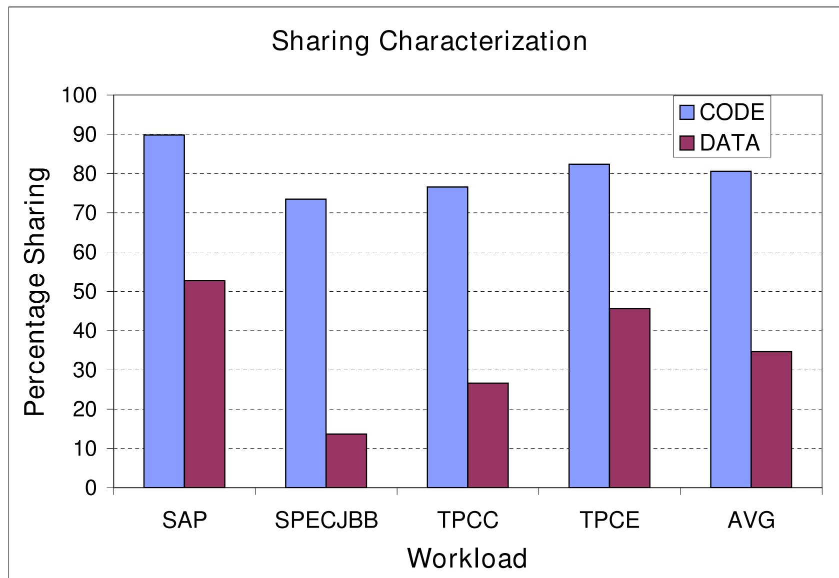


Figure 5.3. Workload characterization: sharing trend in server workloads

the workloads. All workloads exhibit high degree of code page sharing. SPECjbb has poor data sharing characteristics and TPC-E and SAP have high data sharing charateristics. We also observed that when threads share code pages, the degree of sharing is usually high. This implies that the "Rp:I + Share4:D" model would have to replicate code pages in most cache banks. However, the working set size of code pages is very small compared to data pages (0.6% on average), but the relative access count of code pages is much higher (57% on average).

5.4.4 Evaluation of Page Coloring Schemes

In order to isolate the performance impact of page-coloring schemes, we study the effect of these schemes as a function of cache capacity. We use IPC and miss-ratio as the performance metrics. We assume a tree-network topology and vary each cache bank's capacity. All numbers are normalized against the 1 MB baseline bank that employs no page coloring (Base-No-PC), *i.e.*, even private pages may be placed in remote banks.

Figure 5.4 shows the performance effect of various page-coloring schemes and Figure 5.5 shows the corresponding miss rates. For cache banks smaller than 2MB,

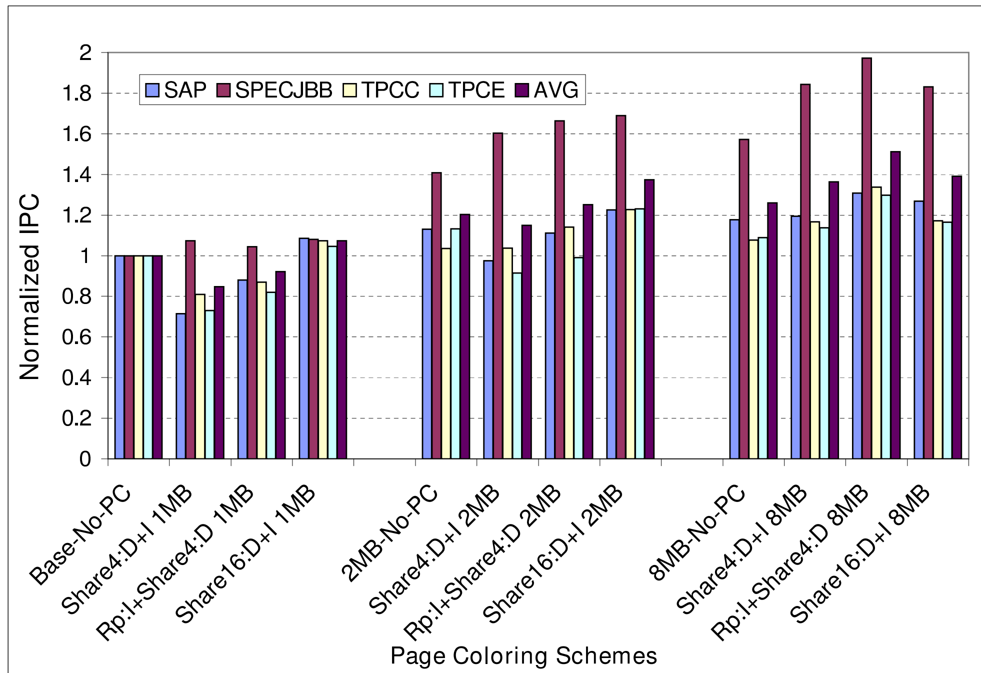


Figure 5.4. Performance evaluation of page coloring schemes as a function of cache capacity

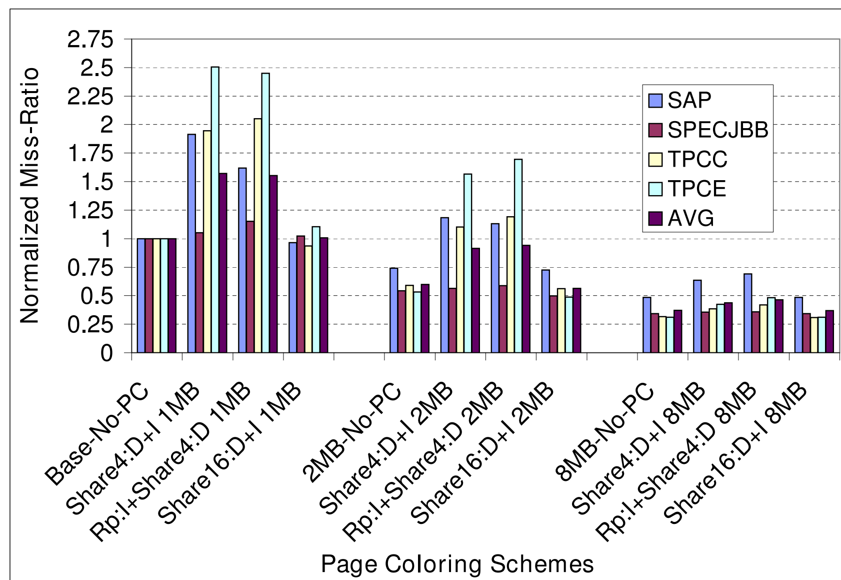


Figure 5.5. Miss ratio of page coloring schemes as a function of cache capacity

Share4:D+I and Rp:I+Share4:D perform worse than Share16:D+I and no page-coloring case. Since Share4:D+I and Rp:I+Share4:D map all shared pages to the central four banks, these banks suffer from high pressure and high miss rates due to less cache capacity. However, since Share16:D+I distributes shared pages across all banks, it does not suffer from bank pressure. We also observe that Share16:D+I performs slightly better (7%) than the base case (no page coloring) as it is able to optimize access latency for private pages for small cache banks.

For a larger cache bank (8MB), Rp:I+Share4:D performs the best compared to all the other schemes. The overall performance improvement in Share4:D compared to no page-coloring (1MB) baseline is 50%. Rp:I+Share4:D has higher performance as it has lower access latency for all the code pages and does not suffer from high bank pressure in spite of code replication overheads due to available cache capacity. We observe Share4:D+I and Share16:D+I to have comparable performance. We notice that performance of Share16:D+I does not scale with cache capacity as much. Thus, when cache capacity is not a constraint, Rp:I+Share4:D delivers the best performance compared to other schemes and makes an ideal candidate for SRAM-DRAM cache.

With regards to workloads, SPECjbb always performs better than the baseline (with no page-coloring) irrespective of the page coloring scheme employed. Since SPECjbb has low degree of application sharing, it does not suffer from pressure on shared banks and performs better due to communication optimization enabled by page-coloring. However, SAP and TPC-E have poor performance due to high bank pressure for small sized banks as they have high degree of sharing. Figure 5.5 further affirms our observations and shows miss ratios for various schemes. Clearly, SPECjbb has low miss ratio compared to other workloads. As expected for most cases, Rp:I+Share4:D has higher miss-ratio due to additional conflicts introduced by code replication.

5.4.5 Reconfigurable SRAM-DRAM Cache

We next evaluate our SRAM-DRAM reconfigurable cache. If an SRAM bank encounters high bank pressure, it enables the DRAM bank directly above. The total available cache capacity in each combined SRAM-DRAM bank is 8.5 MB with 0.5 MB tag space. However, we assume the total per-bank capacity to be 8MB to ease performance modeling. We compare our reconfigurable cache against Base-No-PC and Base-2x-No-PC. Figure 5.6 shows IPC for various configurations and baselines. When the reconfiguration heuristics are enabled, the model is prepended with *R*, and with no dynamic reconfiguration (the DRAM banks are always enabled), the model is prepended with *NR*. We assume 100,000 cycles penalty for flushing the caches during reconfiguration.

We observe that all workloads on average perform 45-62% better than Base-No-PC when DRAM banks are switched on all the time and 20-35% better compared to Base-2x-No-PC depending upon the page-coloring scheme. When compared to Base-3-level, our NR cases perform 11-26% better depending upon the page coloring scheme. Our best case performance improvement with reconfiguration heuristic enabled when compared to Base-3-level is 19%.

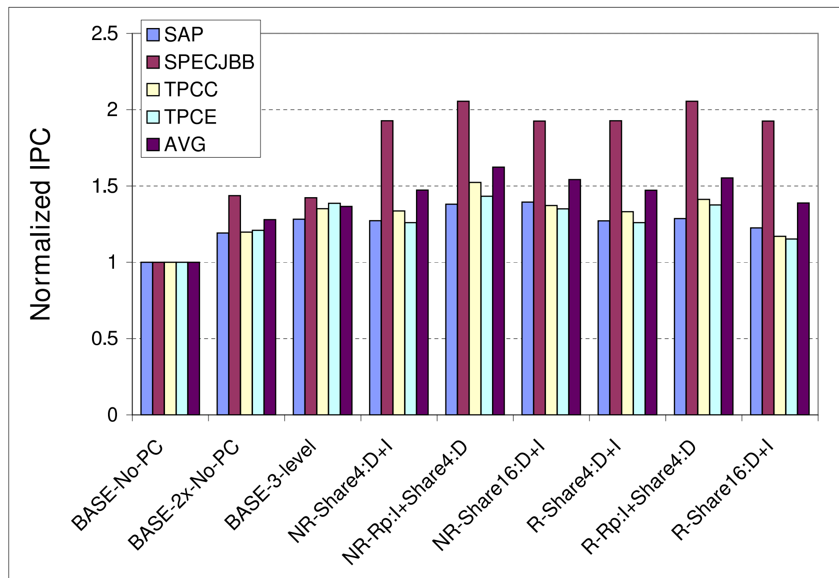


Figure 5.6. Performance evaluation of SRAM-DRAM cache

With our reconfiguration heuristic, we observe that DRAM banks are switched on all the time for central shared banks in the Rp:I+Share4:D case. The noncentral banks in this model and occasionally in Share4:D+I have DRAM switched off due to low bank pressure. Amongst all the page-coloring schemes, Rp:I+Share4:D yields the best performance with and without reconfiguration heuristic. Since we stall all cores during reconfiguration, the overheads of our heuristic lower the overall performance gain compared to the best case performance with DRAMs enabled all the time. However, we expect this overhead to reduce if the workloads run long enough to amortize the cost of reconfiguration.

Figure 5.7 illustrates the average percentage of time DRAM banks are switched off for each of the cache banks for the Rp:I+Share4:D scheme. The DRAM banks are switched off most of the time for noncentral banks but shared central banks have DRAM switched on all the time. Amongst various workloads, we found that due to low sharing behavior in SPECJBB, we have high capacity pressure even on noncentral banks for this workload. Thus, it has its DRAM turned on most of the time for all page-coloring schemes. SAP workload has multiple program phases leading to maximum number of reconfigurations compared to all the workloads. We show the overall bank access distribution for different page coloring schemes

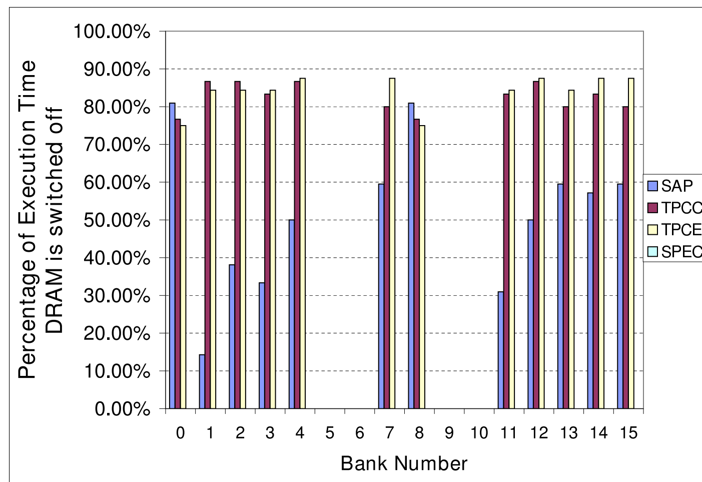


Figure 5.7. Percentage of total time each DRAM bank is switched off for Rp:I+Share4:D. Banks 5, 6, 9, and 10 are the central banks.

in Figure 5.8. Share4:D+I and Rp:I+Share4:D have maximum accesses to shared banks (5,6,9,10) whereas Share16:D+I has accesses distributed evenly amongst all the banks.

Figures 5.9 and 5.10 show the average distribution of hits in SRAM and DRAM ways for all the cache banks for Rp:I+Share4:D schemes with and without reconfiguration, respectively. Without dynamic reconfiguration, the constant enabling of DRAM banks implies that several L2 look-ups are serviced by the slower and energy-inefficient DRAM banks. Dynamic reconfiguration ensures that DRAM banks are looked up only when bank pressure is high (typically only in central banks). Even the 50% of accesses to DRAM ways in these central banks can perhaps be reduced if we employ some MRU-based heuristics to move blocks from DRAM to SRAM ways.

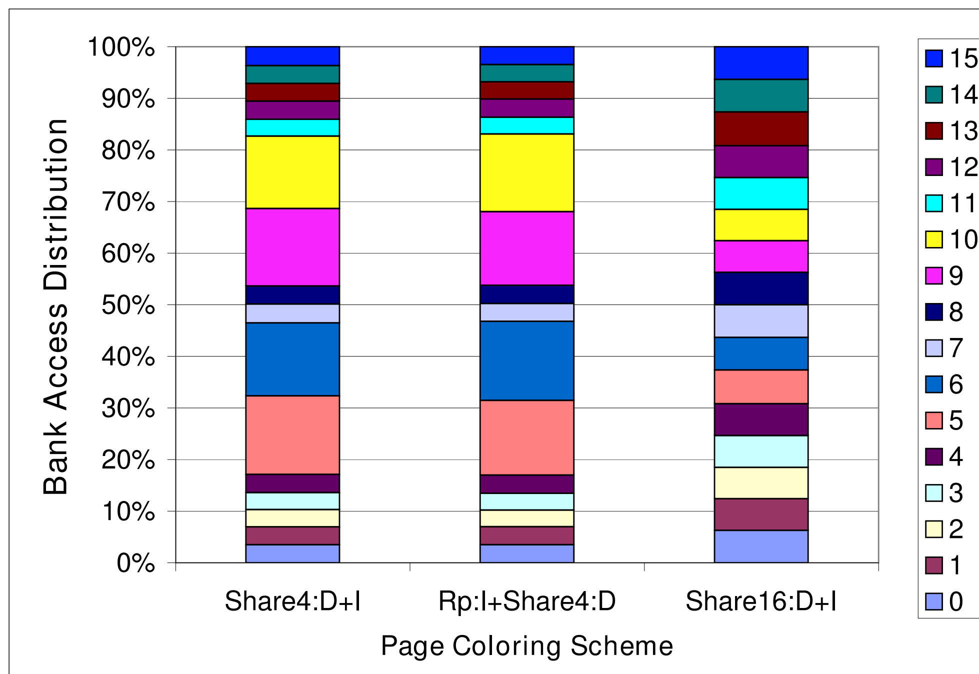


Figure 5.8. Distribution of accesses based on bank number.

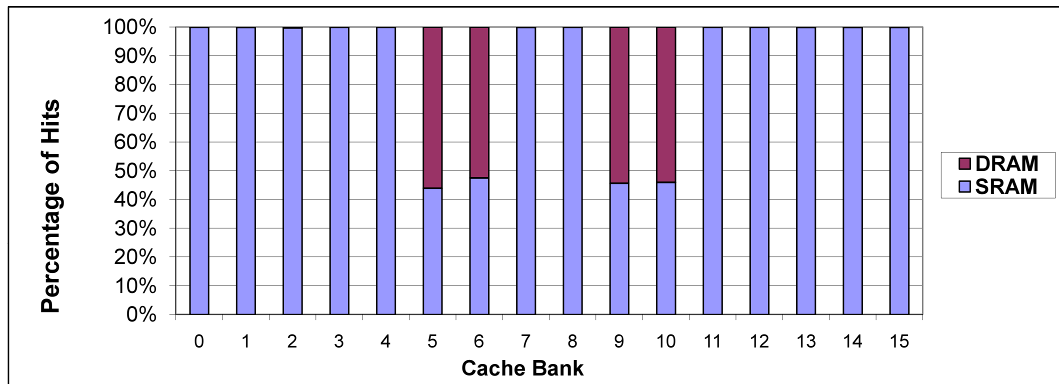


Figure 5.9. Average percentage of hits in SRAM/DRAM ways for R-Rp:I+Share4:D.

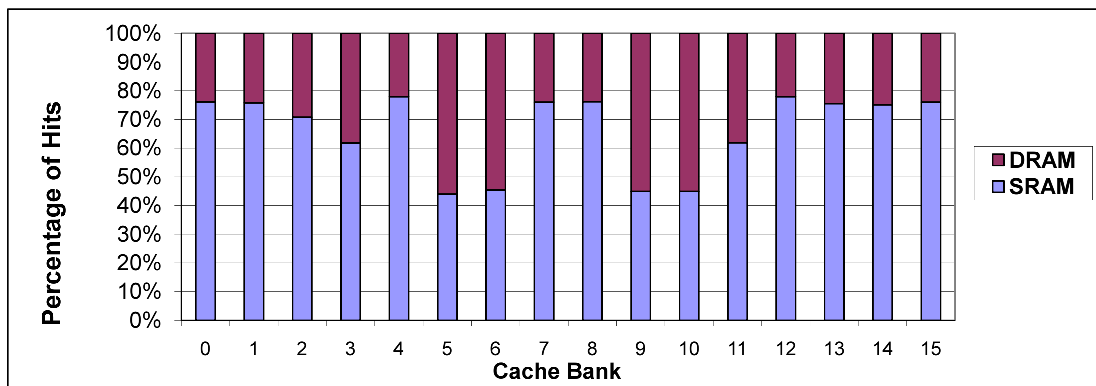


Figure 5.10. Average percentage of hits in SRAM/DRAM ways for NR-Rp:I+Share4:D.

5.4.6 Interconnect Evaluation

We find that our page-coloring schemes optimize the network traffic latency significantly enough that the type of network employed does not influence performance much. We expect the performance improvement to be more significant in network topologies with greater number of cores and also where there is true sharing across all nodes. The tree topology performs 2% better than the grid topology in our 16-core simulations with page-coloring enabled. With higher wire and router delays, we expect the performance impact to be more significant. The use of the tree network does lower overall router power overheads due to fewer routers. We

observe a 48% reduction in overall network power, compared against a baseline with no page coloring and a grid topology.

The following statistics help explain the power improvements. Figure 5.11 demonstrates the bank access distribution in a tree/grid network. *Sibling* refers to a bank that is one hop away on the tree topology, *local* refers to the core's own bank, and *Distant* refers to communication with all other banks. We observe that the baseline with no page coloring sends most of the requests to distant banks and Rp:I+Share4:D maximizes local requests due to code replication.

We also carried out an initial analysis of thermal behavior with Hotspot 4.1 [60] augmented with 3D parameters [33]. Consistent with other papers on 3D stacking of SRAM and DRAM, we observed an increase in temperature of about 7 ℃.

5.5 Summary

The central idea in this proposed work is the design of a cache that enables easy reconfigurability across multiple dies. This central idea is placed in the context of two related essential elements of large cache organization (OS-based page coloring

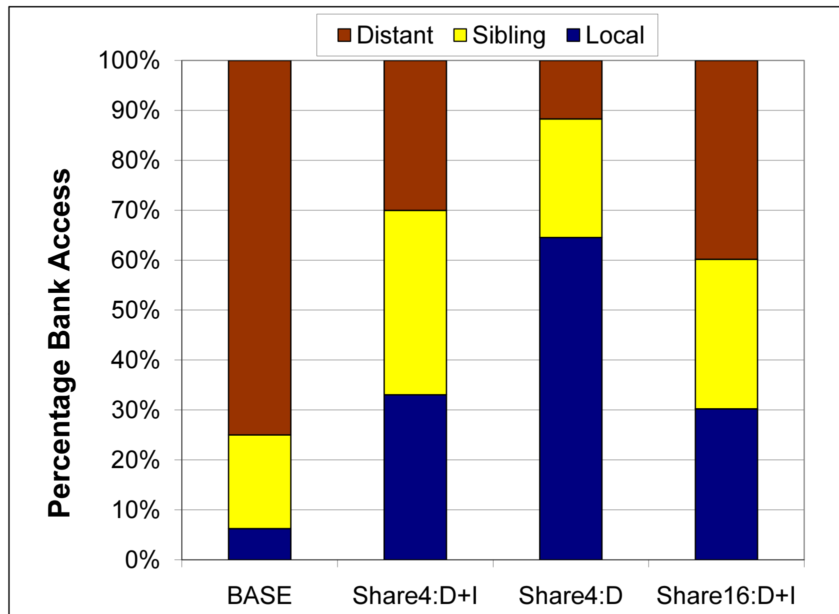


Figure 5.11. Distribution of bank accesses based on distance

and on-chip network design). OS-based page coloring not only introduces variation in bank working sets, it also enforces a very specific traffic pattern on the network. We take advantage of 3D technology to implement a heterogeneous reconfigurable cache that addresses the problem of variable working sets by selectively providing high capacity at low cost. We advocate a tree topology for the on-chip network because (i) the use of page coloring and S-NUCA avoids search across all banks, (ii) page coloring forces requests to be serviced by local nodes or radiate in/out of central banks, and (iii) reconfiguration prevents a bank from having to spill its data into neighboring banks. The proposed reconfigurable cache model improves performance by up to 19% over the best baseline, along with 48% savings in network power. Our results show that commercial workloads exercise variable pressures on cache banks after page coloring, requiring a dynamic enabling/disabling of DRAM space. The performance of these workloads is very sensitive to cache capacity and quick access to instruction (code) pages. This requires smart page coloring mechanisms as well as the large L2 capacity afforded by dynamic enabling of DRAM banks. Thus, in this chapter, we propose an efficient cache hierarchy that takes advantage of die-heterogeneity enabled by 3D stacking to design a hybrid SRAM-DRAM cache for capacity management. Such a heterogeneous cache is able to benefit from capacity enabled by DRAM technology and superior power/delay properties of SRAM technology.

CHAPTER 6

CONCLUSIONS

In this dissertation, we study some of the major challenges faced by the semiconductor industry, specifically reliability and wire delay. We explore techniques that take advantage of emerging 3D die stacking technology and propose novel solutions that help alleviate the above-mentioned issues. We especially highlight the importance of stacking of heterogeneous dies enabled by 3D integration and propose compelling applications from a computer architect's perspective. The first proposal advocates the use of additional die for stacking redundant hardware in an error-resilient older process. We also explore a complexity-effective checker core architecture that has been optimized for power-efficiency. We show that such a power-efficient reliable architecture is suitable for 3D implementation. The second proposal deals with the design of a hybrid SRAM-DRAM cache that combines the advantages of both SRAM and DRAM technologies by stacking DRAM die above SRAM die.

Chapter 3 explores the design-space for reducing power overheads of redundancy in redundantly multithreaded architectures. Our proposal of using an in-order checker core makes a compelling application for future heterogeneous CMPs. We demonstrate that by investing in more intercore bandwidth by enabling register value prediction in an in-order checker core coupled with dynamic frequency scaling, we can lower the overall power overheads of redundancy to up to 10%. We show using analytical models that workload parallelization and dynamic voltage scaling hold little promise.

In Chapter 3, our research was conducted for a heterogeneous system where the leading core is an aggressive out-of-order core. Future CMPs are expected to be

a mix of simple and complex leading cores as shown by some of the projections in [108]. We believe that for such a scenario, there are several ways to utilize our in-order checker core model. If the leading core is a simple in-order core, we can execute the corresponding trailing thread on a simple trailing core. The performance optimization enabled by a perfect data cache, branch predictor and register value prediction would hold true even in this case. However, the relative power savings due to redundancy would be much lower as both cores are inherently power-efficient. We may even be able to map multiple trailing threads on a single wide-issue multithreaded trailing core. In a heterogeneous CMP design space, if there are varying number of simple and complex cores, then our model would need to support dynamic coupling between cores instead of one-to-one mapping. This would require significant redesign especially of interconnects and buffering mechanisms.

In Chapter 4, we propose a 3D reliable processor organization and evaluate many of its design considerations. A checker core on the upper die helps reduce interconnect lengths and part of the upper die can also be employed to increase L2 cache size. The isolation of the checker core to a separate die reduces the impact on the leading core's layout, wiring, and cycle time. The snap-on functionality allows customers the option to detect and recover from failures, while not impacting the manufacturing cost for customers that are less concerned about reliability. The most significant impact of this design is a temperature increase of up to 7 ℃ or alternatively, a performance loss of up to 8% if a constant thermal constraint is imposed. The impact on temperature and performance is much less if we assume a simple low-power microarchitecture for the in-order core. We also show that the checker core is more error-resilient because it typically operates at a much lower frequency. Error tolerance can be further increased by implementing the upper die on an older process that suffers less from dynamic timing errors and soft errors. The move to an older technology increases power consumption, but reduces temperature because the power density of the hottest block is lowered. This helps limit the overhead of the checker core to a 3 ℃ temperature increase or a 4%

performance loss. Our results largely argue in favor of heterogeneous dies for this specific application domain of low-overhead redundancy.

In Chapter 5, we propose the design of a cache that enables easy reconfigurability across multiple dies. This central idea is placed in the context of two related essential elements of large cache organization (OS-based page coloring and on-chip network design). OS-based page coloring not only introduces variation in bank working sets, it also enforces a very specific traffic pattern on the network. We take advantage of 3D technology to implement a heterogeneous reconfigurable cache that addresses the problem of variable working sets by selectively providing high capacity at low cost. We advocate a tree topology for the on-chip network because (i) the use of page coloring and S-NUCA avoids search across all banks, (ii) page coloring forces requests to be serviced by local nodes or radiate in/out of central banks, and (iii) reconfiguration prevents a bank from having to spill its data into neighboring banks. The proposed reconfigurable cache model improves performance by up to 19% over the best baseline, along with 48% savings in network power. Our results show that commercial workloads exercise variable pressures on cache banks after page coloring, requiring a dynamic enabling/disabling of DRAM space. The performance of these workloads is very sensitive to cache capacity and quick access to instruction (code) pages. This requires smart page coloring mechanisms as well as the large L2 capacity afforded by dynamic enabling of DRAM banks.

Overall our novel research in the area of mixed-technology stacking has demonstrated compelling applications that can take advantage of this property of 3D die stacking. When 3D technology becomes matured, we can make two design choices if given the option of stacking additional dies based on our proposals. We can either implement a checker core and SRAM banks in an older process to reap reliability benefits or we can design the hybrid SRAM-DRAM cache to improve the performance of commercial workloads.

Although, each of these different research studies have explored architectures for certain process nodes based on which process node was current at the time of

our evaluation, we believe that our results and conclusions will hold true for any relative heterogeneous process nodes.

6.1 Future Work

As future work, we would like to explore other application avenues that can take advantage of die heterogeneity enabled by 3D technology. Some of the possible directions include exploring quantum devices and SOI process modules for stacking over traditional CMOS based processors. Our view is that such heterogeneous systems can balance the pros and cons of any given technology. Other interesting avenues include the stacking of flash storage and its implications on the memory hierarchy design.

We would like to extend our study in Chapter 3 to focus on the thermal impact of redundant hardware. As the degree of redundancy increases, the contribution of redundant threads to total system power also increases. In such a setting, it may be worth studying how the power consumed by in-order cores can be further reduced. We would also like to explore other interesting applications of an in-order checker core in heterogeneous CMPs such as postsilicon verification. For example, a single trailing core can suffice for performing postsilicon verification of multiple cores in a CMP. Other interesting application can be to enable the design of an aggressive leading core that has less conservative design margins. Such a leading core can always utilize a simple checker core for detecting timing violations related errors.

Chapter 4 highlights the stacking of redundant hardware on an older process. While our research focused mainly on the checker core stacking, we would like to explore this further by considering stacking of other redundant hardware such as checkpointed register files. Since many reliable processors checkpoint their state for recovery, implementing this checkpoint hardware in older process can protect the recovery data further from reliability failures.

In Chapter 5, we proposed the design of a hybrid reconfigurable SRAM-DRAM cache. We can improve our work in reconfiguration heuristics further by consid-

ering energy/power metrics. We would also like to evaluate other workloads including multiprogrammed and parallel emerging workloads. Ways to maximize SRAM access in our proposed SRAM-DRAM cache is also another possible future direction of this proposal.

REFERENCES

[1] N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookreson, A. Vo, S. Mitra, and J. Maiz, "Radiation Induced Soft Error Rates of Advanced CMOS Bulk Devices," in *Proceedings of IEEE International Reliability Physics Symposium*, 2006.

[2] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *Proceedings of the 40th International Symposium on Microarchitecture (MICRO-40)*, December 2007.

[3] Semiconductor Industry Association, "International Technology Roadmap for Semiconductors 2007." `http://www.itrs.net/Links/2005ITRS/Home2007.htm`.

[4] N. Muralimanohar and R. Balasubramonian, "Interconnect Design Considerations for Large NUCA Caches," in *Proceedings of the 34th International Symposium on Computer Architecture (ISCA-34)*, June 2007.

[5] S. Mukherjee, J. Emer, and S. Reinhardt, "The Soft Error Problem: An Architectural Perspective," in *Proceedings of HPCA-11 (Industrial Session)*, February 2005.

[6] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinatorial Logic," in *Proceedings of DSN*, June 2002.

[7] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The Case for Lifetime Reliability-Aware Microprocessors," in *Proceedings of ISCA-31*, June 2004.

[8] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, "Interconnect Power Dissipation in a Microprocessor," in *Proceedings of System Level Interconnect Prediction*, February 2004.

[9] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. Cook, "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors," *IEEE Micro*, November/December 2000.

[10] R. I. Bahar and S. Manne, "Power and Energy Reduction Via Pipeline Balancing," in *Proceedings of ISCA-28*, pp. 218–229, July 2001.

[11] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy Caches: Simple Techniques for Reducing Leakage Power," in *Proceedings of ISCA-29*, May 2002.

[12] Y. Xie, G. Loh, B. Black, and K. Bernstein, "Design Space Exploration for 3D Architectures," *ACM Journal of Emerging Technologies in Computing Systems*, vol. 2(2), pp. 65–103, April 2006.

[13] Samsung Electronics Corporation, "Samsung Electronics Develops World's First Eight-Die Multi-Chip Package for Multimedia Cell Phones," 2005. (Press release from `http://www.samsung.com`).

[14] Tezzaron Semiconductor. `www.tezzaron.com`.

[15] J. Rattner, "Predicting the Future," 2005. Keynote at Intel Developer Forum (article at `http://www.anandtech.com/tradeshows/showdoc.aspx?i=2367&p=3`).

[16] S. Mysore, B. Agrawal, N. Srivastava, S. Lin, K. Banerjee, and T. Sherwood, "Introspective 3D Chips," in *Proceedings of ASPLOS-XII*, October 2006.

[17] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. M. Sule, M. Steer, and P. D. Franzon, "Demystifying 3D ICs: The Pros and Cons of Going Vertical," *IEEE Design and Test*, vol. 22(6), pp. 498–510, November 2005.

[18] S. Gupta, M. Hilbert, S. Hong, and R. Patti, "Techniques for Producing 3D ICs with High-Density Interconnect," in *Proceedings of the 21st International VLSI Multilevel Interconnection Conference*, September 2004.

[19] G. Loi, B. Agrawal, N. Srivastava, S. Lin, T. Sherwood, and K. Banerjee, "A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy," in *Proceedings of DAC-43*, June 2006.

[20] B. Black, M. Annavaram, E. Brekelbaum, J. DeVale, L. Jiang, G. Loh, D. McCauley, P. Morrow, D. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die Stacking (3D) Microarchitecture," in *Proceedings of MICRO-39*, December 2006.

[21] J. Cong, A. Jagannathan, Y. Ma, G. Reinman, J. Wei, and Y. Zhang, "An Automated Design Flow for 3D Microarchitecture Evaluation," in *Proceedings of ASP-DAC*, January 2006.

[22] K. Puttaswamy and G. Loh, "Thermal Analysis of a 3D Die-Stacked High-Performance Microprocessor," in *Proceedings of GLSVLSI*, April 2006.

[23] K. Puttaswamy and G. Loh, "Implementing Register Files for High-Performance Microprocessors in a Die-Stacked (3D) Technology," in *Proceedings of ISVLSI*, March 2006.

[24] K. Puttaswamy and G. Loh, "The Impact of 3-Dimensional Integration on the Design of Arithmetic Units," in *Proceedings of ISCAS*, May 2006.

[25] K. Puttaswamy and G. Loh, "Thermal Herding: Microarchitecture Techniques for Controlling HotSpots in High-Performance 3D-Integrated Processors," in *Proceedings of HPCA*, February 2007.

[26] J. Kim, C. Nicopoulos, D. Park, R. Das, Y. Xie, N. Vijaykrishnan, and C. Das, "A Novel Dimensionally-Decomposed Router for On-Chip Communication in 3D Architectures," in *Proceedings of ISCA-34*, June 2007.

[27] D. Park, S. Eachempati, R. Das, A. Mishra, Y. X. N. Vijaykrishnan, and C. Das, "MIRA : A Multilayered Interconnect Router Architecture," in *Proceedings of ISCA-35*, June 2008.

[28] N. Madan and R. Balasubramonian, "Leveraging 3D Technology for Improved Reliability," in *Proceedings of the 40th International Symposium on Microarchitecture (MICRO-40)*, December 2007.

[29] W. Zhang and T. Li, "Microarchitecture Soft Error Vulnerability Characterization and Mitigation under 3D Integration Technology," in *Proceedings of MICRO-41*, December 2008.

[30] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, N. Vijaykrishnan, and M. Kandemir, "Design and Management of 3D Chip Multiprocessors Using Network-in-Memory," in *Proceedings of ISCA-33*, June 2006.

[31] T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, S. Reinhardt, K. Flautner, and T. Mudge, "PicoServer: Using 3D Stacking Technology to Enable a Compact Energy Efficient Chip Multiprocessor," in *Proceedings of ASPLOS-XII*, October 2006.

[32] C. C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari, " Bridging the Processor-Memory Performance Gap with 3D IC Technology," *IEEE Design and Test of Computers*, vol. 22, pp. 556–564, November 2005.

[33] G. Loh, "3D-Stacked Memory Architectures for Multi-Core Processors," in *Proceedings of ISCA-35*, June 2008.

[34] T. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," in *Proceedings of MICRO-32*, November 1999.

[35] M. Gomaa, C. Scarbrough, and T. Vijaykumar, "Transient-Fault Recovery for Chip Multiprocessors," in *Proceedings of ISCA-30*, June 2003.

[36] S. Mukherjee, M. Kontz, and S. Reinhardt, "Detailed Design and Implementation of Redundant Multithreading Alternatives," in *Proceedings of ISCA-29*, May 2002.

[37] M. Rashid, E. Tan, M. Huang, and D. Albonesi, "Exploiting Coarse-Grain Verification Parallelism for Power-Efficient Fault Tolerance," in *Proceedings of PACT-14*, 2005.

[38] S. Reinhardt and S. Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," in *Proceedings of ISCA-27*, pp. 25–36, June 2000.

[39] J. Smolens, J. Kim, J. Hoe, and B. Falsafi, "Efficient Resource Sharing in Concurrent Error Detecting Superscalar Microarchitectures," in *Proceedings of MICRO-37*, December 2004.

[40] T. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient-Fault Recovery via Simultaneous Multithreading," in *Proceedings of ISCA-29*, May 2002.

[41] M. Gomaa and T. N. Vijaykumar, "Opportunistic Transient-Fault Detection," in *Proceedings of ISCA-32*, Jun 2005.

[42] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," in *Proceedings of ISCA-23*, May 1996.

[43] C. McNairy and R. Bhatia, "Montecito: A Dual-Core, Dual-Thread Itanium Processor," *IEEE Micro*, vol. 25(2), March/April 2005.

[44] E. Grochowski, R. Ronen, J. Shen, and H. Wang, "Best of Both Latency and Throughput," in *Proceedings of ICCD-22*, October 2004.

[45] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. Scott, "Energy Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling," in *Proceedings of HPCA-8*, pp. 29–40, February 2002.

[46] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *Proceedings of ISCA-27*, pp. 83–94, June 2000.

[47] D. Burger and T. Austin, "The Simplescalar Toolset, Version 2.0," Tech. Rep. TR-97-1342, University of Wisconsin-Madison, June 1997.

[48] R. Balasubramonian, N. Muralimanohar, K. Ramani, and V. Venkatachalapathy, "Microarchitectural Wire Management for Performance and Power in Partitioned Architectures," in *Proceedings of HPCA-11*, February 2005.

[49] R. Kumar, V. Zyuban, and D. Tullsen, "Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads, and Scaling," in *Proceedings of the 32nd ISCA*, June 2005.

[50] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen, "Single ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction," in *Proceedings of the 36th International Symposium on Micro-Architecture*, December 2003.

[51] J. A. Butts and G. Sohi, "A Static Power Model for Architects," in *Proceedings of MICRO-33*, December 2000.

[52] S. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads," in *Proceedings of ICCAD*, 2002.

[53] L. Clark, "Circuit Design of XScale Microprocessors," in *Symposium on VLSI Circuits (Short Course on Physical Design for Low-Power and High-Performance Microprocessor Circuits)*, June 2001.

[54] J. Ray, J. Hoe, and B. Falsafi, "Dual Use of Superscalar Datapath for Transient-Fault Detection and Recovery," in *Proceedings of MICRO-34*, December 2001.

[55] E. Rotenberg, "AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors," in *Proceedings of 29th International Symposium on Fault-Tolerant Computing*, June 1999.

[56] N. Wang, J. Quek, T. Rafacz, and S. Patel, "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," in *Proceedings of DSN*, June 2004.

[57] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," in *Proceedings of MICRO-36*, December 2003.

[58] S. Kumar and A. Aggarwal, "Reduced Resource Redundancy for Concurrent Error Detection Techniques in High Performance Microprocessors," in *Proceedings of HPCA-12*, Feb 2006.

[59] S. Kumar and A. Aggarwal, "Self-checking Instructions - Reducing Instruction Redundancy for Concurrent Error Detection," in *Proceedings of PACT*, Sep 2006.

[60] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, and K. Sankaranarayanan, "Temperature-Aware Microarchitecture," in *Proceedings of ISCA-30*, pp. 2–13, 2003.

[61] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: A Power-Performance Simulator for Interconnection Networks," in *Proceedings of MICRO-35*, November 2002.

[62] D. Tarjan, S. Thoziyoor, and N. Jouppi, "CACTI 4.0," Tech. Rep. HPL-2006-86, HP Laboratories, 2006.

[63] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 1st ed., 2003.

[64] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *Proceedings of ASPLOS-X*, October 2002.

[65] K. Krewell, "Sun's Niagara Pours on the Cores," *Microprocessor Report*, vol. 18, pp. 11–13, September 2004.

[66] H. de Vries, "Die area scaling factor." (`http://www.chip-architect.com/news/2007_02_19_Various_Images.html`).

[67] L. Hsu, R. Iyer, S. Makineni, S. Reinhardt, and D. Newell, "Exploring the Cache Design Space for Large Scale CMPs," in *Proceedings of dasCMP*, November 2005.

[68] S. Das, A. Fan, K.-N. Chen, and C. Tan, "Technology, Performance, and Computer-Aided Design of Three-Dimensional Integrated Circuits," in *Proceedings of International Symposium on Physical Design*, April 2004.

[69] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. Carter, "Interconnect-Aware Coherence Protocols for Chip Multiprocessors," in *Proceedings of 33rd International Symposium on Computer Architecture (ISCA-33)*, pp. 339–350, June 2006.

[70] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P. Strenski, and P. Emma, "Optimizing Pipelines for Power and Performance," in *Proceedings of MICRO-35*, November 2002.

[71] N. Madan and R. Balasubramonian, "Power Efficient Approaches to Redundant Multithreading," *IEEE Transactions on Parallel and Distributed Systems (Special issue on CMP architectures)*, vol. Vol.18, No.8, August 2007.

[72] K. Puttaswamy and G. Loh, "Implementing Caches in a 3D Technology for High Performance Processors," in *Proceedings of ICCD*, October 2005.

[73] P. Reed, G. Yeung, and B. Black, "Design Aspects of a Microprocessor Data Cache using 3D Die Interconnect Technology," in *Proceedings of International Conference on Integrated Circuit Design and Technology*, May 2005.

[74] Y.-F. Tsai, Y. Xie, N. Vijaykrishnan, and M. Irwin, "Three-Dimensional Cache Design Using 3DCacti," in *Proceedings of ICCD*, October 2005.

[75] B. Beckmann, M. Marty, and D. Wood, "ASR: Adaptive Selective Replication for CMP Caches," in *Proceedings of MICRO-39*, December 2006.

[76] J. Chang and G. Sohi, "Co-Operative Caching for Chip Multiprocessors," in *Proceedings of ISCA-33*, June 2006.

[77] Z. Chishti, M. Powell, and T. Vijaykumar, "Optimizing Replication, Communication, and Capacity Allocation in CMPs," in *Proceedings of ISCA-32*, June 2005.

[78] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. Keckler, "A NUCA Substrate for Flexible CMP Cache Sharing," in *Proceedings of ICS-19*, June 2005.

[79] E. Speight, H. Shafi, L. Zhang, and R. Rajamony, "Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors," in *Proceedings of ISCA-32*, June 2005.

[80] M. Zhang and K. Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," in *Proceedings of ISCA-32*, June 2005.

[81] P. Kundu, "On-Die Interconnects for Next Generation CMPs," in *Workshop on On- and Off-Chip Interconnection Networks for Multicore Systems (OCIN)*, December 2006.

[82] H. S. Wang, L. S. Peh, and S. Malik, "A Power Model for Routers: Modeling Alpha 21364 and InfiniBand Routers," in *IEEE Micro, Vol 24, No 1*, January 2003.

[83] W. Dally, "Workshop on On- and Off-Chip Interconnection Networks for Multicore Systems (OCIN)," 2006. Workshop program and report at `http://www.ece.ucdavis.edu/~ocin06/`.

[84] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS," in *Proceedings of ISSCC*, February 2007.

[85] C. Kim, D. Burger, and S. Keckler, "An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches," in *Proceedings of ASPLOS-X*, October 2002.

[86] B. Beckmann and D. Wood, "Managing Wire Delay in Large Chip-Multiprocessor Caches," in *Proceedings of MICRO-37*, December 2004.

[87] Z. Chishti, M. Powell, and T. Vijaykumar, "Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures," in *Proceedings of MICRO-36*, December 2003.

[88] S. Cho and L. Jin, "Managing Distributed, Shared L2 Caches through OS-Level Page Allocation," in *Proceedings of MICRO-39*, December 2006.

[89] M. Awasthi, K. Sudan, and R. Balasubramonian, "Dynamic Hardware-Assisted Software-Controlled Page Placement to Manage Capacity Allocation and Sharing within Large Caches," in *Proceedings of HPCA-15*, Feb 2009.

[90] H. Dybdahl and P. Stenstrom, "An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors," in *Proceedings of HPCA-13*, February 2007.

[91] Y. Jin, E. J. Kim, and K. H. Yum, "A Domain-Specific On-Chip Network Design for Large Scale Cache Systems," in *Proceedings of HPCA-13*, February 2007.

[92] B. Beckmann and D. Wood, "TLC: Transmission Line Caches," in *Proceedings of MICRO-36*, December 2003.

[93] Z. Guz, I. Keidar, A. Kolodny, and U. Weiser, "Nahalal: Memory Organization for Chip Multiprocessors," *IEEE Computer Architecture Letters*, vol. vol.6(1), May 2007.

[94] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "A Dynamically Tunable Memory Hierarchy," *IEEE Transactions on Computers*, vol. 52(10), pp. 1243–1258, October 2003.

[95] P. Ranganathan, S. Adve, and N. Jouppi, "Reconfigurable caches and their application to media processing," *Proceedings of ISCA-27*, pp. 214–224, June 2000.

[96] C. Zhang, F. Vahid, and W. Najjar, "A Highly Configurable Cache Architecture for Embedded Systems," in *Proceedings of ISCA-30*, June 2003.

[97] R. Iyer, L. Zhao, F. Guo, R. Illikkal, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt, "QoS Policies and Architecture for Cache/Memory in CMP Platforms," in *Proceedings of SIGMETRICS*, June 2007.

[98] M. Qureshi and Y. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," in *Proceedings of MICRO-39*, December 2006.

[99] K. Varadarajan, S. Nandy, V. Sharda, A. Bharadwaj, R. Iyer, S. Makineni, and D. Newell, "Molecular Caches: A Caching Structure for Dynamic Creation of Application-Specific Heterogeneous Cache Regions," in *Proceedings of MICRO-39*, December 2006.

[100] M. Tremblay and S. Chaudhry, "A Third-Generation 65nm 16-Core 32-Thread Plus 32-Scout-Thread CMT," in *Proceedings of ISSCC*, Febuary 2008.

[101] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies," in *Proceedings of ISCA-35*, June 2008.

[102] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures," in *Proceedings of MICRO-33*, pp. 245–257, December 2000.

[103] R. Balasubramonian, S. Dwarkadas, and D. Albonesi, "Dynamically Managing the Communication-Parallelism Trade-Off in Future Clustered Processors," in *Proceedings of ISCA-30*, pp. 275–286, June 2003.

[104] A. Dhodapkar and J. E. Smith, "Managing Multi-Configurable Hardware via Dynamic Working Set Analysis," in *Proceedings of ISCA-29*, pp. 233–244, May 2002.

[105] E. Duesterwald, C. Cascaval, and S. Dwarkadas, "Characterizing and Predicting Program Behavior and its Variability," in *Proceedings of PACT-12*, September 2003.

[106] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The Alpha 21364 Network Architecture," in *IEEE Micro*, vol. 22, 2002.

[107] L. Zhao, R. Iyer, J. Moses, R. Illikkal, S. Makineni, and D. Newell, "Exploring Large-Scale CMP Architectures Using ManySim," *IEEE Micro*, vol. 27, no. 4, pp. 21–33, 2007.

[108] M. Hill and M. Marty, "Amdahl's law in the multicore era," *IEEE Computer*, July 2008.