Scaling High-Performance Interconnect Architectures to Many-Core Systems

by

Korey LaMar Sewell

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2012

Doctoral Committee:

Professor Trevor Mudge, Chair
Professor Scott Mahlke
Professor Dennis Sylvester
Adjunct Associate Professor Steven K. Reinhardt

To my wife Shetia and my son Kaden

ACKNOWLEDGEMENTS

First off, I would like to thank God for putting me in the position that I am in today. As any Ph.D student knows, there are many times where the academic grind and struggle of completing a doctoral degree seems overwhelming. Thus, I have to thank God for putting the right people in my life to keep me going and giving me the peace of mind of knowing that He put me at Michigan for a reason. My faith was truly a key factor in persevering and persisting through this process.

Next, I have to say thank you to my two advisors during my time at Michigan. Professor Trevor Mudge and Professor Steve Reinhardt were instrumental in facilitating my development as a student, allowing me to explore interesting research projects, and most importantly supporting me through life. It was refreshing and invaluable to have advisors who supported my ideas and challenged me to prove whether ideas were worthwhile or not. My favorite Steve quote is "It's necessarily complex..." while my favorite Trev quotes are: "We've been doing it for years...It's called computers!" and "(insert lab member's name here), You've got to think!"

I have to thank all my labmates for being a great support system. The "m5/gem5" team– Nate Binkert, Ali Saidi, Ron Dreslinski, Kevin Lim, Lisa Hsu, and Gabe Black–introduced me to a level of software and simulator development that can only be referenced as "awesome". I learned so much from you all and I thank you for helping me build a skill set that will serve me well into the future. Geoff Blake, Tony Gutierrez, Joe Pusderis, Tom Manville, Michael Cieslak, Reetu Das, Kyla McMullen, Joe Greathouse, Andrea Pellegrini, Jason Clemons, Ben Cassell, Timur Alperovich, Dave Meisner and Steve Pelley are

among the many other people who were always around for great research conversation, good laughs, and overall just a great environment.

I also have to thank my best friends at Michigan for helping me have a great experience outside of just “lab-life”. My 1st summer at Michigan, Jason Torrey and I visited just about every inch of Ann Arbor. Thanks for showing me the ropes Jason! Kevin Carter was my long-time roommate at Michigan and has been an inspiration to me. Watching him complete his Ph.D showed me that there was a path to having a great life inside and outside of school. Cedric Armand introduced me to golf, snowboarding, and what “New York” life is in Michigan. I can’t say enough about Ced’s “you only have 1 life, live it!” attitude. It’s contagious. Bradley Campbell was and always will be good for a worthy debate. We’ve “fought” hours at a time over topics ranging from basketball (Kobe Bryant/LeBron James!) to food (chiclet and tacos?!) to politics. “Chuck” Sutton is also a guy that’s been a true friend: always available when you need him and definitely always available when there is a good time to be had! All of the aforementioned guys were like brothers to me at Michigan and always provided encouragement when I needed it.

Of course, I have to thank my mother, Gertrude Green, and my father, William Sewell, for all the books, Saturday library outings, and relentless “haggling” about school over the years. My little brother, Kenneth Wise, has always made me feel like I could solve the world’s problems. Although that’s obviously impossible, I appreciate his consistent emotional support. These three people might be the only ones more excited then me about this Ph.D!

Last *but* certainly not least, I have to thank my wife Shetia and my son Kaden for being there for me every night and providing a daily reminder of what I was truly fighting for in this Ph.D process. There’s nothing like the love and support of your family to keep you going when things get tough.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF FIGURES viii

LIST OF ABBREVIATIONS xiii

ABSTRACT xiv

CHAPTER

I. Introduction 1

1.1 Trends in Performance Scaling 2
1.1.1 Single-Core Performance Scaling 2
1.1.2 The Multi-Core Era 7
1.1.3 Scaling from Multi-Core to Many-Core 8
1.2 Scalability and the Impact of On-Chip Interconnects 9
1.2.1 Busses 9
1.2.2 Crossbars 9
1.2.3 Network-On-Chip 10
1.3 Contributions 12
1.4 Organization 13

II. Background and Related Work 14

2.1 Interconnect Characteristics 14
2.1.1 Units of Communication 15
2.1.2 Metrics 16
2.1.3 Arbitration 18
2.1.4 Limitations 20
2.2 Busses 20
2.3 Crossbars 22

2.4 Network-on-Chip . . . 24
2.4.1 Topologies . . . 25
2.4.2 Router Microarchitecture . . . 28
2.4.3 Routing . . . 30
2.5 Related Works . . . 31

III. Swizzle-Switch: A High-Radix, Self-Arbitrating Crossbar . . . 33

3.1 Overview . . . 34
3.2 Layout and Data-Transmission Phase . . . 36
3.3 Arbitration Phase . . . 37
3.3.1 Arbitration Mechanism . . . 38
3.3.2 Least Recently Granted . . . 40
3.3.3 Most Recently Granted (MRG) . . . 42
3.3.4 Selective LRG and MRG . . . 42
3.3.5 Priority Swap and Reversal . . . 44
3.3.6 Round Robin . . . 45
3.3.7 QoS Arbitration . . . 45
3.4 Silicon Validation . . . 46
3.5 Enhanced 32nm Design and Analysis . . . 47
3.6 Conclusions . . . 48

IV. Swizzle-Switch Networks for Many-Core Systems . . . 50

4.1 Interconnect Design Components . . . 51
4.2 Swizzle Switch Network . . . 52
4.2.1 Coherence Protocol . . . 53
4.2.2 Timing and Layout Evaluation . . . 54
4.2.3 Reliability . . . 55
4.3 Mesh Topology . . . 55
4.4 Flattened Butterfly . . . 57
4.5 Simulation Methodology . . . 58
4.6 Performance Analysis . . . 58
4.7 Energy and Power Analysis . . . 62
4.8 Sensitivity Analysis . . . 62
4.8.1 Router Pipelines . . . 63
4.8.2 Virtual Channels . . . 63
4.8.3 Interconnect Frequency . . . 64
4.8.4 Out-of-Order Cores . . . 65
4.9 Conclusions . . . 66

V. Scalable 3D Interconnects . . . 68

5.1 3D Integration Technology . . . 69
5.2 3D-Swizzle-Switch Networks . . . 70

5.2.1 Architecture 70
5.2.2 Performance Results 72
5.2.3 Thermal Analysis 72
5.3 XPoint: Scaling Many-Core Busses to 3D 73
5.3.1 Architecture 74
5.3.2 Methodology 76
5.3.3 Performance Analysis 79
5.4 Conclusions 81

VI. Conclusions 83

BIBLIOGRAPHY 86

LIST OF FIGURES

Figure

1.1 Processor Frequency Trends in Intel Processors [95]. 3

1.2 Reported Microprocessor Power Trends [51]. 5

1.3 Types of Multithreaded Processors [110]. This example assumes a 4-wide superscalar processor. Each vertical set of boxes represents one execution cycle and different colors represent different threads. 6

1.4 QoS in network-on-chip. Figures (a) and (b): All nodes generate traffic directed to a hotspot located at (8,8) with injection rate of 0.05 flit/cycle/node, and the bar graphs show the accepted service rate per source node for (a) 8x8 mesh and (b) 64-radix SSN. Figures (c) and (d) demonstrate the same effect for uniform random traffic at injection rate of 1 flit/cycle/node. Unfairness metric derived from [25]. 10

1.5 Comparison of interconnect power of SSN and Mesh Network-on-Chip . 11

2.1 High-level view of an interconnection network. In this example, core (C_0) and memory (M_0) components are connected to the interconnect fabric through their own communication channel. 15

2.2 Breakdown of a 64-byte interconnect message into 8 packets and 4 flits. . 16

2.3 Offered Traffic v. Message Latency in a Interconnection Network. 17

2.4 A bus interconnect connecting cores to memory. 21

2.5 High-level view of a (a) 8x8 crossbar interconnect and an (b) 4x4 matrix crossbar connecting cores to memory. A separate crossbar connecting memory to cores would be needed to complete the circuit in (a). 23

2.6 A Mesh Network-on-Chip Topology. 24

2.7 Common on-chip network topologies. Illustration from [52]. 26

2.8 A 2-ary, 3-fly butterfly network. Illustration from [52]. 27

2.9 The Microarchitecture for a credit-based, virtual channel router found in a Mesh Topology (note: 5 inputs and 5 outputs). The circled numbers represent router pipeline stages. 28

3.1 High level view of (a) bus interleaving required for mux-based crossbars; (b) a traditional matrix style crossbar with arbiter/controller consuming space and requiring additional input wires; (c) the proposed *Swizzle-Switch* design that reuses input/output busses for programming/arbitration of the crossbar with the arbitration logic placed under the dense metal interconnect. 34

3.2 Scaling trends of the *Swizzle-Switch* Interconnect in 32nm vs. a conventional crossbar (Simulated). 35

3.3 Circuit implementation of the *Swizzle-Switch* Interconnect. Each output *column* in the interconnect uses the same request bit from each input bus. Each input *row* uses the same bit from each output bus to perform arbitration. The expanded view of the crosspoint shows the stored configuration and crosspoint connections for each bit. It also shows the programming of priority bits using the output bus. Because the crosspoint is for Input Row 0, the arbitration sense amp is on *output wire 0*. Similarly, because it is Output column 1, the request line is drawn from *input wire 1*. 36

3.4 Conceptual example of *Swizzle-Switch* Arbitration. A matrix represents one complete output column. Each output column arbitrates and transmits data independently of other output columns. 38

3.5 A 5x3 crossbar showing arbitration circuits. Each output column arbitrates and transmits independently. 39

3.6 Detailed blowup of arbitration for the K^{th} output column of a 5-input *Swizzle-Switch* interconnect. 40

3.7 Least-Recently-Granted priority update of a *Swizzle-Switch* output. . . . 41

3.8 (a) 64-Core *Swizzle-Switch Network* System (b) Maximum Request Latency for Random and Round-Robin Arbitration Policies (Normalized to LRG). 41

3.9 Most Recently Granted priority update of a *Swizzle-Switch* output. 42

3.10 Selective LRG (a) and MRG (b) priority update of a *Swizzle-Switch* output. 43

3.11 Priority Swap (a) and Reversal (b) of a *Swizzle-Switch* output. 44

3.12 A Quality-of-Service Circuit for the *Swizzle-Switch*. 46

3.13 Die Photo of the 45nm silicon test chip. 47

3.14 Measured Frequency and Bandwidth Efficiency of the silicon test chip from 3.13. 47

3.15 Bandwidth and Speed of a *Swizzle-Switch* with 128-bit busses in 32nm. Both repeated and non-repeated versions are presented. When using repeaters the *Swizzle-Switch* scales to designs as large as 128x128x128 resulting in 15 Tbps of total bandwidth. 48

4.1 (a) High-level architecture diagram (a) of the 64-core *Swizzle-Switch Network* (SSN) built with *Swizzle-Switch* crossbars. (b) The floor-plan of the (SSN) system and estimated dimensions. Octants are colored to aid the reader in seeing how wires leave the crossbar. The total chip area is $204mm^2$, each core/L1 tile consumes $0.74mm^2$, the L2 tiles consume $4.5mm^2$ and the *Swizzle-Switch* consumes $6.65mm^2$. 51

4.2 (a) Classification of communication messages required for coherence (b) Wiring diagram for combining three *Swizzle-Switches* into a 64×64×128bit crossbar. The wires are labeled by the quadrant to which they connect. Each wire in the diagram represents either 3, 5, or 8 busses, where each bus is 128-bits. The overall area of the Crossbar is $6.65mm^2$($\sim 4\%$ of the 64 tile system). 53

4.3 Floor-plan of the Mesh and Flattened Butterfly systems with estimated dimensions. The total size of both chips is $190mm^2$. 56

4.4 Cycle Analysis for 64 core Mesh, FBFly, and SSN topologies during parallel regions of the SPLASH2 benchmarks. 60

4.5 Histogram of L1 cache miss latency for the Radix benchmark. 60

4.6 Total interconnect power (top) broken down by components within the Mesh, FBFly, and SSN systems for all benchmarks tested. Total system energy (bottom) for each benchmark broken down by component. Overall the SSN reduces interconnect power by 33% over the Mesh and 28% over the FBFly on average. As a result of the lower interconnect power and better performance the total SSN system energy is 25% less than the Mesh and 11% less than the FBFly. 61

4.7 Sensitivity analysis using ideal, 2-stage speculative routers. Histogram of the L1 miss latencies for the Radix benchmark. 63

4.8 Mesh sensitivity to the number of virtual channels (VCs) per virtual network for the Raytrace benchmark. For this example, there is only a 1 % performance improvement using 5 VCs (over 3 VCs) per virtual network. The enlarged data point represents the configuration used in Section 4.6. . 64

4.9 Sensitivity to the interconnect frequency for the Mesh and SSN (Cores remain at 1.5GHz). Results show that a Mesh w/4-cycle routers needs to be run at 4x the frequency of a SSN to achieve the same performance. The enlarged data points for the SSN and Mesh represent the configurations used in Section 4.6. 65

4.10 Speedups of a 64-core SSN using out-of-order cores over 64-core NoCs also using out-of-order cores. Benchmarks shown represent the 3 traffic classes referenced in Section 4.6. The compute intensive benchmark (WaterNSquared) sees a 1.31x improvement while the memory-intensive (Radix) and synchronization-sensitive workloads see $\sim$2x and $\sim$3x improvements respectively when using out-of-order cores. 66

5.1 Top level view of the Centip3De 7-layer 3D system [33] built on Tezzaron 3D stacking technology and a cross section of the same process on the 3D-MAPS system [59]. Note the TSV's are only 6.47 microns deep and the wafer is thinned to less than 12 microns which is important for reducing thermal resistance and RC delays. 69

5.2 (a) A 3D *Swizzle-Switch Network* achieved by stacking four 2D *Swizzle-Switch Network* layers and using TSV's to interconnect the layers. (b) The modified circuits for the 3D *Swizzle-Switch Network*. Even numbered outputs are arbitrated on the top layer, odd outputs are arbitrated on the bottom layer. Input request lines must be forwarded from the top→bottom or bottom→top through TSV connections. 71

5.3 Speedup of the 3D-SSN on 2-layer and 4-layer systems compared to a 2D-SSN. The benchmarks most sensitive to interconnect delay are plotted as well as the average across all benchmarks. 72

5.4 HotSpot simulation of 64 Core SSN system on 1 and 4 layers for the worst case benchmark. The peak temperature of the 3D chip is 60^o Celsius. 73

5.5 High level view of (a) a conventional bus based architecture, and (b) The *XPoint-2D* architecture. Caches in a vertical column are all assigned to the same address range. No snooping is required between vertical columns. The horizontal connections use point-to-point links. 74

5.6 Diagram of the *XPoint 3D* design. 75

5.7 Runtime (solid lines) Bus Utilization (dotted lines) *vs*. core counts for Conventional Bus and XPoint Systems. A straight line for runtime represents ideal scaling of the benchmark. 78

5.8 Speedup comparison for Bus, *XPoint 2D*, and *XPoint 3D* interconnects. The best performing parameters for each benchmark and configuration are used. Details of number of cores, slices, and layers are found in Table 5.4 79

LIST OF ABBREVIATIONS

CGMT Coarse-Grained Multithreading

CMP Chip Multiprocessor

CPU Central Processing Unit

FBFly Flattened Butterfly

FGMT Fine-Grained Multithreading

I/O Input/Output

LRG Least-Recently Granted

MRG Most-Recently Granted

NoC Network-on-Chip

NUMA Non-Uniform Memory-Access

QoS Quality-of-Service

RR Round-Robin

SMT Simultaneous Multithreading

SSN *Swizzle-Switch Network*

TDMA Time Division Multiple Access

TSV Through-Silicon Via

VC Virtual Channel

ABSTRACT

Scaling High-Performance Interconnect Architectures to Many-Core Systems

by

Korey LaMar Sewell

Chair: Trevor Mudge

The ever-increasing demand for performance scaling has made multi-core (2-8 cores) chips prevalent in today's computing systems and foreshadows the shift toward many-core (10s-100s of cores) chips in the near future. Although the potential performance gains from many-core systems remain appealing, the widespread adoption of these systems hinges on their ability to scale performance while simultaneously satisfying Quality-of-Service (QoS) and energy-efficiency constraints.

This work makes the case that the interconnect for these many-core systems has a significant impact on the aforementioned scalability issues. The impact of interconnects on many-core systems is illustrated by observing that the degree of the interconnect has a signicant effect on system scalability and demonstrating that the architecture of high-radix, many-core systems are feasible, energy-efficient, and high-performance.

The feasibility of high-radix crossbars for many-core systems is first shown through a new circuit-level building block called the *Swizzle-Switch*. A 32nm *Swizzle-Switch* utilizes integrated arbitration techniques to provides an energy- and area-efficient switching element that improves the scalability of crossbars to a high radices. The *Swizzle-Switch* is shown to operate at frequencies up to 1.5GHz for 128-bit, radix-64 crossbars

and also to have the ability to implement many arbitration policies such as Least-Recently Granted (LRG) and Round-Robin (RR). Results show that *Swizzle-Switch*'s LRG arbitration policy reduces the worst-case request access latency by 1.83× and 2.03× on average over round robin and random arbitration schemes, respectively.

This work then shows how a many-core system called the *Swizzle-Switch Network* can use the *Swizzle-Switch* as the central building block for a flat crossbar interconnect. The *Swizzle-Switch Network* is shown to be advantageous to traditional Network-on-Chip (NoC) for systems up to 64 cores. The *Swizzle-Switch Network* improves system performance by 21%, reduces L1 on-chip average miss latency by 2.2×, and decreases the standard deviation of that L1 miss latency by 3.0× relative to a Mesh NoC topology. Additionally, all of these performance benefits are obtained while providing a 25% energy savings over the Mesh.

The *Swizzle-Switch* is also leveraged as a building block for high-radix NoC topologies that can support many-core architectures. The *Swizzle-Switch*-based Flattened Butterfly topology is demonstrated to provide a 15% speedup, 1.76× smaller L1 on-chip average miss latency, 2.5× reduction in miss latency standard deviation, and 10% energy savings over the Mesh topology.

Finally, the impact that 3D stacking technology has on many-core scalability is evaluated and shown to assist crossbar and bus interconnects in scaling past their traditional limitations. A 3D-optimized *Swizzle-Switch Network* is able to leverage frequency gains to achieve a 15-28% speedup over a 2D-*Swizzle-Switch Network* when using memory-intensive benchmarks. Additionally, a bus-based 64-core architecture is shown to provide an average speedup of 49× over a baseline uniprocessor system when using 3D technology.

CHAPTER I

Introduction

The ever-increasing demand for computing power has driven the computer industry for decades. Significant compute resources are no longer reserved for servers and supercomputers but also for an array of mobile devices such as laptops, tablets, and cell phones. While these compute elements continue to become more powerful, they have reached the point where physical limitations threaten the future scalability of high-performance processors. Consequently, computer architects face the continuing challenge of scaling performance while simultaneously managing area, frequency, and power constraints.

The first part of this chapter details processor scaling trends that have caused the computer systems industry to move from multi-core to many-core systems. Performance scaling is first discussed in the context of single-thread processor designs and then extended to consider pertinent issues for multithreaded processors. After outlining the tradeoffs of multithreaded designs, scaling trends for multi-core and many-core processors are detailed.

The second part of this chapter motivates the need for scalable interconnect architectures. In particular, this section discusses the inability of busses and crossbars to scale for many-core systems and notes that Network-On-Chips have consequently become the default interconnect for many-core designs.

The final part of this chapter outlines the contributions this work makes in analyzing and designing interconnect architectures for many-core systems.

1.1 Trends in Performance Scaling

For years, performance scaling in computer chips has followed the path defined by Moore's law [82, 83]. Moore's prediction that the number of transistors on a chip would double every 18 months has created a market in which single-chip performance improves even while the actual size of a chip decreases. This increase in transistors per chip has coincided with an increase in area for performance enhancing hardware (*e.g.*, large on-chip caches, out-of-order processing, floating point units, etc.). Additionally, computer architects could scale performance simply by waiting for smaller technology nodes to increase processor clock frequency. Rapidly multiplying transistor counts eventually allowed for multiple threads per core, multiple cores on chip, and a host of hybrid technologies to increase performance.

1.1.1 Single-Core Performance Scaling

Early contributions in performance scaling have focused on speeding up a single core (also referred to as CPU) on a single chip. This scaling has primarily consisted of CPU pipeline enhancements enabled by additional Moore's law transistors as well as circuit improvements that have seen single core frequency boosted from the MHz to the GHz range.

1.1.1.1 Frequency Scaling

Along with Moore's law, Dennard scaling [29] has been a key trend in the performance scaling of a single CPU. Dennard scaling provided that decreasing the size of a transistor was enough to provide an improvement in circuit delay and also consume less power. Figure 1.1 shows an example of the significant frequency advances that have been achieved according to predicted scaling trends. An Intel i386 processor [94] in 1987 operated around the 16MHz range while the 2000 version of the Pentium 4 [48] operated around 1.5GHz. This yearly doubling of clock frequency has been a large contributor to single core perfor-

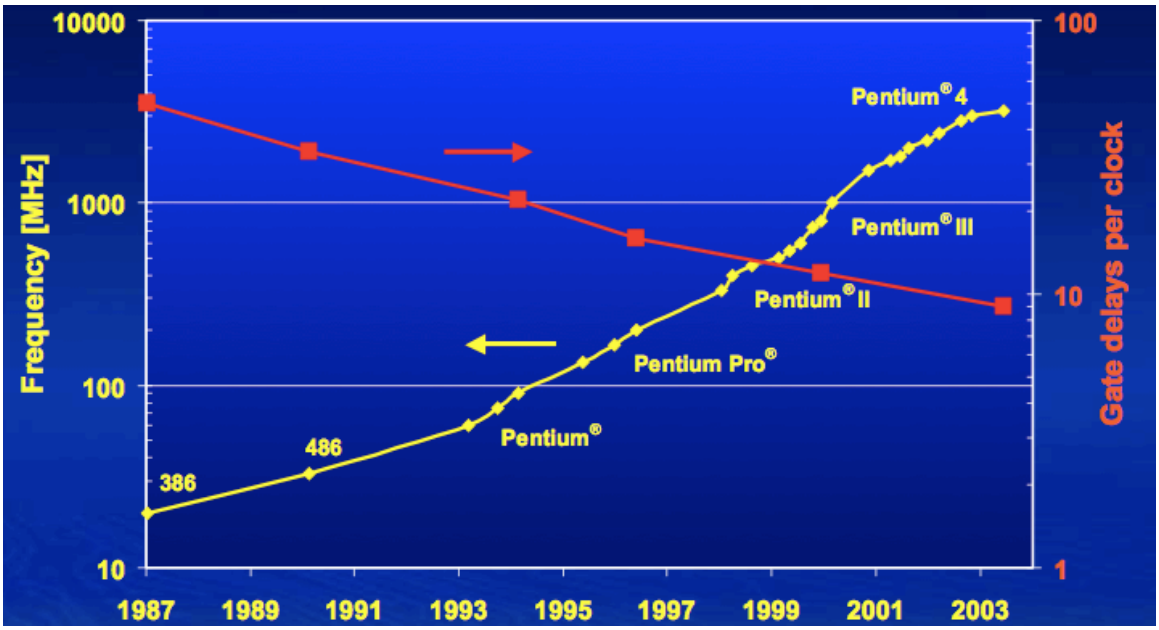


Figure 1.1: Processor Frequency Trends in Intel Processors [95].

mance scaling as application speedup could be achieved without necessarily optimizing the application itself.

Unfortunately, the benefits of Dennard scaling have dissipated past the 130nm technology node. While circuit advances have still allowed technology nodes to continue scaling past 32nm [65], the proportional relationship of power and frequency ($P = CV^2f$) makes it difficult to blindly increase clock frequency without suffering significant thermal disadvantages.

Sutter [109] analyzes transistor, clock frequency, and power trends for Intel processors produced through the year 2009 and observes that the speed of Intel CPUs peaked around 3.4GHz due to cooling constraints. Consequently, today's chip designers can no longer count on the speedup of improved technology nodes to arbitrarily improve clock frequency.

1.1.1.2 Single-Thread Scaling

The ability to scale single-thread performance has historically been one of the great beneficiaries of Moore's law. Many authors have proposed performance-enhancing hardware to pipeline, speculate, and cache single-threaded applications. The increased presence of on-chip caches has also been a direct beneficiaries of Moore's law. Whereas early processors might have used a single level of caching, it is common for current processors to use up to three levels of cache at sizes approaching 8 to 16 MB. The larger amount of on-chip caching continues to aid single-thread scaling by decreasing average memory access latency and in turn providing more CPU cycles for a single-thread.

In-order pipelines, like the MIPS4K [81], divide the execution of an instruction into distinct stages (*e.g.*, fetch, decode, execute, memory, and writeback). This division of tasks provides greater instruction throughput and also decreases the minimum cycle time of the CPU (as the maximum frequency of the pipeline is limited by the pipeline stage with the longest delay). The Prescott derivatives of the Intel Pentium 4 series [119] have used up to 31 pipeline stages in an effort to maximize how fast the CPU can process instructions.

Other single-thread scaling techniques, such as out-of-order processing, attempt to leverage the Instruction Level Parallelism (ILP) within an application to improve performance. While scoreboards [44] can be used to track and execute independent instructions, register renaming techniques are the primary way modern processors exploit ILP. Tomasolu's reservation-station approach [112] is recognized as the seminal work in out-of-order processing and has been followed by designs such as the MIPS 10K [124] which use free lists, load-store queues, and physical register files to further optimize register renaming.

Processors like the Intel Itanium [80] take advantage of ILP using Very Long Instruction Word (VLIW) architectures. These techniques rely on the compiler to expose the parallelism for the hardware in the form of multiple instructions to execute on one cycle (VLIW).

The continued scaling of ILP hardware has become limited by the available ILP in

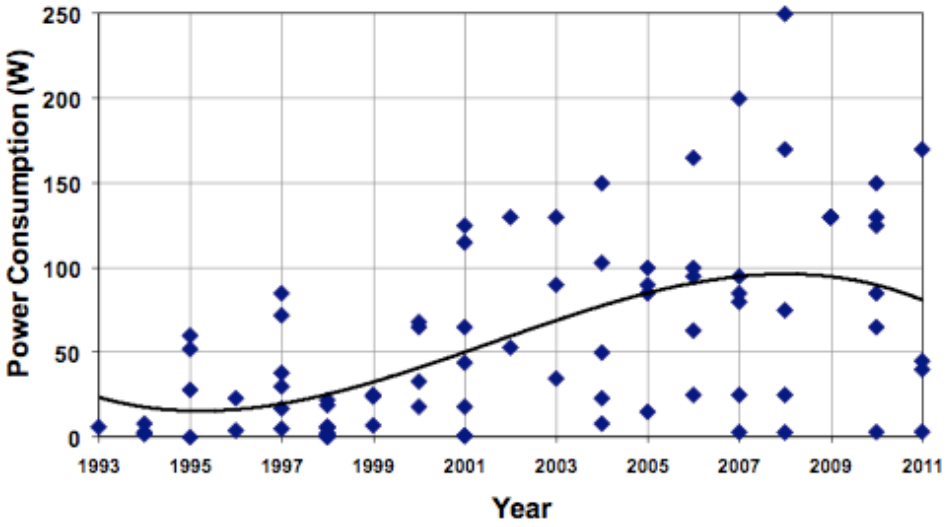


Figure 1.2: Reported Microprocessor Power Trends [51].

applications as well as the cost of out-of-order hardware. While application-specific parallelism can not be helped without changes in the software, exploiting the maximum amount of parallelism entails increasing expensive hardware structures such as the instruction window (queue), number of renaming registers, reorder buffers and load-store queues. In particular, the large buffers needed to implement hardware such as the instruction window often use power-intensive, content-associative memories (CAM), which can also negatively impact the maximum processor frequency if on the processor pipeline's critical path. Figure 1.2 shows processor power trends over the past decade. Because the cost of cooling a chip is prohibitive, single-chip power has stalled in the 10s of Watts range. Consequently, recent work has shown that there is only a marginal amount gain left to obtain in single-thread hardware without incurring a disproportionate amount of power overhead [6].

1.1.1.3 Exploiting Multithreaded Performance

As a response to the problems of single-core scaling, computer architects turned toward multithreaded architectures for increased processing power. While the latency to complete a single-thread of execution only continues to improve incrementally, the opportunity

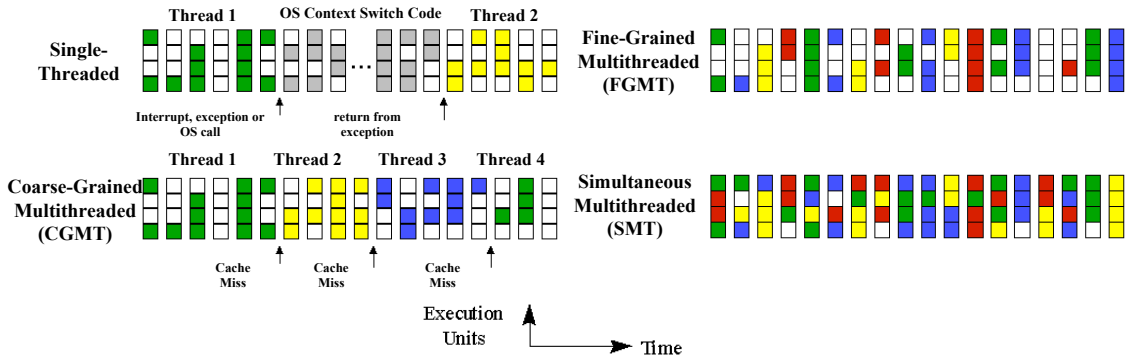


Figure 1.3: Types of Multithreaded Processors [110]. This example assumes a 4-wide superscalar processor. Each vertical set of boxes represents one execution cycle and different colors represent different threads.

to increase throughput amongst multiple threads has given rise to numerous architectures striving to exploit thread-level parallelism (TLP) in addition to ILP.

Multithreaded architectures can be split up into three types: Coarse-Grained Multithreading (CGMT) , Fine-Grained Multithreading (FGMT) and Simultaneous Multithreading (SMT). Figure 1.3 illustrates multithreaded architecture types. CGMT designs switch threads on every long-latency CPU event such as a cache miss or page fault. While one thread is servicing this type of event, the CPU switches threads instead of stalling. FGMT architectures switch threads at a much smaller time quantum. Thread switch can be triggered every cycle, using some priority scheme (*e.g.*, round-robin), or reacting to some speculative event (*e.g.*, branch instruction).

SMT has been the most popular of multithreading architectures. First introduced by Tullsen *et al.* [115], SMT architectures process multiple hardware threads in parallel by dynamically partitioning resources amongst threads (*e.g.*, register file, instruction queue, etc.) The Intel Pentium 4's version of SMT, HyperThreading [123], was among the 1st commercially available SMT processors and allowed up to 2 hardware threads at one time. Prior studies have explored fetch policies [114], partitioning schemes [93, 17], and using SMT threads as helpers (*e.g.*, prefetchers) for a main running thread [18]. Works such as Oehmke's Virtual Context Architecture [87] and Sewell's EXtreme Virtual Pipelining [105]

propose SMT processors that can support high thread counts. These works optimize the limitation of the static storage resources within the processor pipeline (*e.g.*, the register file) to create virtual hardware contexts. Burns and Gaudiot [16] find that 8-way (threaded) SMT designs can provide a 5x throughput improvement over a traditional out-of-order processor.

Despite the throughput gains, Burns' work has also shown that single-core multithreaded processors are limited by the amount of inter-thread resource sharing (*e.g.*, the cache) and the hardware cost of adding additional thread contexts in the system. The additional area overhead of an 8-way SMT processor is found to be 46% compared to a non-SMT, out-of-order design. Thus, recent SMT processors have been limited to about 4 threads per system while in-order, SMT designs like the SPARC Niagara [63] have much less thread state to replicate and have implemented CPUs with 8-16 hardware threads per core.

1.1.2 The Multi-Core Era

As transistor and voltage scaling started to see diminishing performance returns for both single- and multi-threaded CPUs, chip architects began to leverage increased transistor counts to build chips with multiple CPUs (cores) on one die. Commonly referred to as Chip Multiprocessor (CMP), these multi-core systems typically replicate the core and private caches for each processing element while sharing a second-level cache and then connecting to a main memory controller. In comparison to a multithreaded processor, replicating per-core resources instead of per-thread resources becomes more energy-efficient as the number of hardware contexts (threads) increases. This is primarily due to the cache thrashing and microarchitectural resource sharing issues that can hinder the performance of multithreaded CPUs with a high number of hardware threads.

In 2001, IBM's Power4 processor [111] realized 2 cores on chip and is regarded as the first commercial, general purpose CMP. The Power4 was quickly followed by designs such as the AMD Opteron [55] and the Intel Itanium 2 [80]. CMPs have now become the

predominant choice for throughput scaling in both the general-purpose and embedded systems markets. Currently, Intel's Core i7 [50] series targets the general-purpose computing market and offers up to 6 cores per processor while the ARM Cortex A9 is a popular choice in the embedded world and can support up to 4 cores on die.

Because single-core multithreading and multi-core systems are not mutually exclusive designs, Chip Multithreading (CMT) architectures like the Sun UltraSparc IV (codename: Niagara [63]) have combined both technologies to maximize throughput. The latest version of the Niagara series (Niagara 3/UltraSparc T3) has 16 inorder cores and 8 threads per cores allowing for the processing of 128 threads simultaneously.

1.1.3 Scaling from Multi-Core to Many-Core

The continued scaling from multi-core (4-16 cores) to many-core (64+ cores) has become prohibited by factors such as power distribution and interconnect topology. The challenge of powering a many-core chip can lead to designs that have "dark silicon" [32]: chips where parts of the area must be turned off due to power constraints. Although dynamic Voltage and Frequency Scaling [104] technology allows architects to tune core frequency at runtime, there is a significant amount of core management necessary to maintain high performance for designs where the frequency is mitigated by the total core count.

Additionally, the on-chip interconnect has become a limiting factor for future many-core chips. Managing communication costs in the form of wire delay, contention, and bandwidth has a large impact on the performance scalability as the number of cores rises. Bus and crossbar-based interconnects are commonplace for small multi-core systems but have been limited by the amount of contention and the high power of their designs respectively. Recently, systems with large core counts have used Network-on-Chip (NoC) topologies such as a ring or mesh to solve the wiring and power problems inherent in traditional bus and crossbar interconnect fabrics. However, enforcing quality of service in NoCs becomes problematic as the core count continues to rise and limit performance gains.

1.2 Scalability and the Impact of On-Chip Interconnects

1.2.1 Busses

The emergence of many-core designs has led to a renewed interest in interconnect techniques because intra-chip communication bottlenecks can compromise performance. Although most commercial multi-core designs have used bus-based communication [49, 4], long bus wires as well as saturated bus contention has hindered the scalability of busses past 8-16 cores [23]. As such, it is clear that bus-based interconnects are not suitable for many-core systems.

1.2.2 Crossbars

To provide better bandwidth, early multi-core systems transitioned from bus-based interconnect fabrics to crossbars [54, 3, 78, 127]. Crossbar-based architectures, like those in the Niagara2 [54] and IBM BlueGene/Q [43], can provide the uniform memory access latency that is unachievable in multi-stage NoC systems. Additionally, crossbar systems provide higher bisection bandwidth and lower complexity solutions for quality-of-service guarantees than NoC designs.

Despite these advantages, large crossbars are generally considered infeasible because the area and power of traditional matrix-style crossbars grow quadratically with crossbar radix. The increased power consumption and die area from high-radix crossbars has led researchers to explore alternatives to flat crossbar topologies [126]. In systems whose core counts approach 64-128, several studies have noted that creating a crossbar to interconnect all cores would cause the interconnect's power and area to dominate the system [66]. Consequently, there has been a paradigm shift towards packet-switched, on-chip networks with regular topologies such as meshes [120, 45] and rings [40, 103].

However, this thesis revisits the ability of crossbars to scale to many-core systems. The integrated arbitration technology found in the *Swizzle-Switch* (detailed in Chapter III) opti-

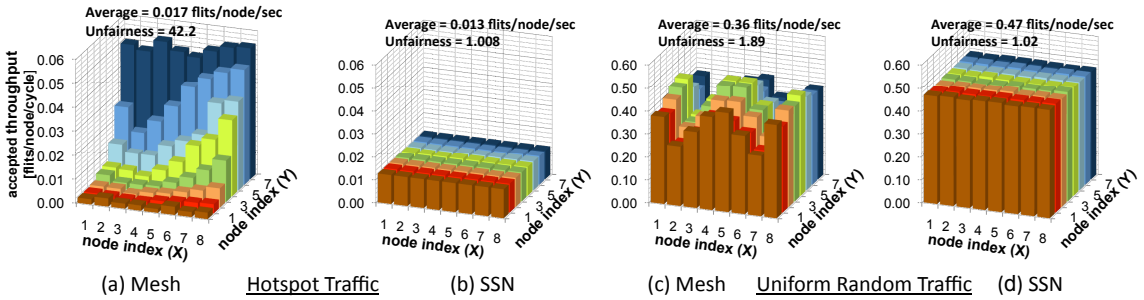


Figure 1.4: QoS in network-on-chip. Figures (a) and (b): All nodes generate traffic directed to a hotspot located at (8,8) with injection rate of 0.05 flit/cycle/node, and the bar graphs show the accepted service rate per source node for (a) 8x8 mesh and (b) 64-radix SSN. Figures (c) and (d) demonstrate the same effect for uniform random traffic at injection rate of 1 flit/cycle/node. Unfairness metric derived from [25].

mizes area and power relative to traditional crossbar systems. This works finds that high-radix crossbars are feasible designs and architects a many-core system called the *Swizzle-Switch Network* (detailed in Chapter IV) to evaluate the tradeoffs against Network-On-Chip systems.

1.2.3 Network-On-Chip

Network-on-Chip (NoC) designs have been advocated as an alternative to bus-based architectures. NoC systems, such as the Tilera Tile64 [120], utilize a distributed multi-stage interconnect design to avoid the scaling issues of long wires. However, this improved scalability comes at the expense of high variability in memory access latencies as well as increased design complexity to guarantee correctness and fairness (*e.g.*, avoiding deadlock, livelock, starvation, etc.).

This shift from a flat interconnect model to non-uniform, multi-hop interconnects has come at the cost of Non-Uniform Cache Access (NUCA) latencies [58]. The ability to provide uniform latency makes using a crossbar an appealing option because predictable latencies remove the need for complex techniques for routing algorithms [91], quality-of-service [39], congestion management [71], data placement [77], and thread scheduling [46].

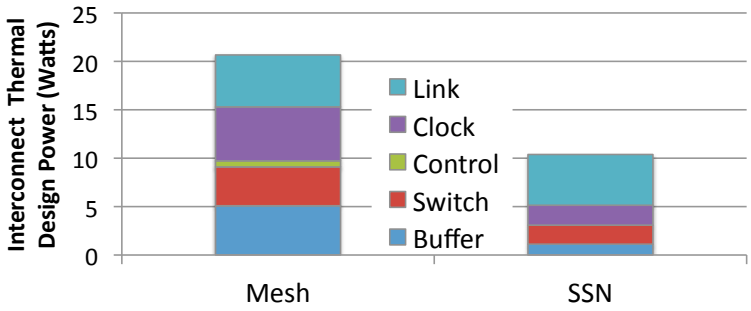


Figure 1.5: Comparison of interconnect power of SSN and Mesh Network-on-Chip

Figure 1.4 shows the high variability of multi-hop, on-chip networks compared to the single-hop *Swizzle-Switch Network* (SSN) (both the Mesh and SSN are assumed to run at the same frequencies in this experiment). In a Mesh network, the accepted throughput at any given node is highly dependent upon the location of the destination node. Under hotspot traffic, nodes closest to destination ($node_{8,8}$) receive the highest throughput, while nodes closest to the center (*e.g.*, $node_{4,4}$) receive the highest throughput when traffic is uniformly distributed. In contrast, the SSN distributes its throughput evenly amongst its nodes allowing for it to see a 40× and 87% fairness improvement for hotspot and uniform random traffic respectively. While there are many research papers that address fairness issues in NoCs, these solutions typically involve complex mechanisms [39, 71] to enforce fairness whereas crossbar topologies simplifies these quality of service concerns by construction.

Additionally, the reduced wiring complexity of a NoC system is bought at a price other than non-uniform latency: collisions can occur within the network. To resolve these collisions and avoid protocol deadlocks, on-chip networks usually require additional buffers per router node (*e.g.*, to provide multiple protocol lanes or virtual channels), which consume significant power and area. As a result, Borkar *et al.* [15] predict that many-core NoCs could consume as much as 80 Watts in future systems. In contrast, a flat crossbar is non-blocking and does not require intermediate buffers, reducing power and area overheads.

Previous studies address the power and latency scalability challenges of NoCs by building concentrated and hierarchical topologies to reduce the required number of on-chip

routers [7, 61, 28]. In contrast to those works, the *Swizzle-Switch Network* (SSN) realizes a single-router flat interconnect with a scalable high-radix crossbar design and minimal end-point buffering. Figure 1.5 illustrates the power savings achieved by this work's proposed SSN design for a synthetic benchmark designed to saturate the network (details for this analysis are found in Chapter IV). Additionally, the technology used in the SSN can be used as a building block for future hierarchical NoC topologies.

1.3 Contributions

This thesis analyzes the scalability of many-core systems and proposes interconnect architectures that can assist in solving these scalability challenges. In particular, this work observes that the degree of the interconnect has a significant effect on system scalability and seeks to build high-radix, many-core systems that will scale performance, power, and quality-of-service metrics to core counts of up to 64.

Many-core scalability efforts are first done in this work by revisiting the design of crossbar and high-radix interconnects in light of advances in circuit techniques that significantly improve crossbar scalability. A new circuit-level building block, the *Swizzle-Switch*, utilizes an integration arbitration technique within the crossbar to build a energy- and area-efficient switching element that can improve the scalability of crossbars to high radices. In addition, this work finds that the multicast ability within the *Swizzle-Switch* makes it a good candidate for many-core system design.

After being scaled to 32nm, multiple *Swizzle-Switches* are used to create the *Swizzle-Switch Network*: a flat, cache-coherent crossbar topology supporting many-core systems of up to 64 cores. The *Swizzle-Switch Network* is shown to scale favorably over a traditional Network-on-Chip (NoC) topology (Mesh) for systems of up to 64 cores. Hierarchical, high-radix NoC topologies are also studied and demonstrated to be scalable architecture choices for systems with core counts above 64.

Finally, this work studies the impact that 3D stacking technology has on many-core

scalability and finds that this emerging technology can help crossbar and bus interconnects scale past their traditional limitations.

1.4 Organization

This work is organized as follows: Chapter II specifies background and related work for interconnect design, Chapter III discusses the design and implementation high-radix crossbars, Chapter IV details and compares the *Swizzle-Switch Network* to Mesh and Flattened Butterfly NoC topologies, and Chapter V extends the scalability discussion toward 3D-stacked crossbar and bus architectures. Finally, conclusions and future works are outlined in Chapter VI.

CHAPTER II

Background and Related Work

The text in this chapter presents background concepts and works related to the design of scalable interconnects for many-core systems. First, characteristics and terminology of interconnect architectures are defined. Next, concepts and architectural details for busses, crossbars, and Network-on-Chips (NoCs) are presented. This chapter then concludes by tying this thesis' contribution of scalable interconnect architectures to relevant related works.

2.1 Interconnect Characteristics

Modern computer systems are made up of many components (*e.g*. CPUs, caches, I/O devices, etc.) communicating over an on-chip interconnect. Figure 2.1 shows a high-level view of an interconnection network. The network consists of core and memory components on the edge of the network as well as *channels* that connect those components to the interconnect. Components can be generally classified into either initiator or response types. For example, cores are an initiator type component as it makes requests to other components in the system, while a memory (or cache) would be a response-type component as it only responds to requests. The interconnect can also contain intermediate channels and switching components (*e.g*., NoC routers) to help facilitate on-chip communication.

Ideally, the interconnect choice for any given system would be high speed, low power, and meet any necessary bandwidth requirements. However, desirable interconnect char-

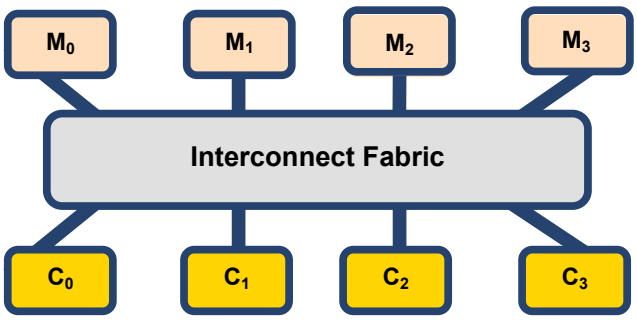


Figure 2.1: High-level view of an interconnection network. In this example, core (C_0) and memory (M_0) components are connected to the interconnect fabric through their own communication channel.

acteristics are often conflicting and force the architect to make tradeoffs in designing the most efficient communication architecture for their particular system. The following text gives an overview of the basic interconnect characteristics and serves as a basis for the terminology used throughout this thesis.

2.1.1 Units of Communication

Messages can be viewed as the logical communication unit between two components in a interconnect network. For example, a memory component will only process once it has received some type of request message from another component within the network. The message would include all the data the memory component would need in order to handle the request correctly (*e.g.*, address, data, etc.).

On-chip networks recognize *packets* as their form of communication unit. Any destination, data, or command attributes contained in a message is first translated into packets before they leave their component and begin transfer over the network. A particular interconnect can choose to implement variable or fixed size length packets. In the case that a message is greater then the size of one packet, the interconnect will split the message up into multiple packets in order to satisfy the message requirements. However, messages and packets are typically the same size in on-chip networks.

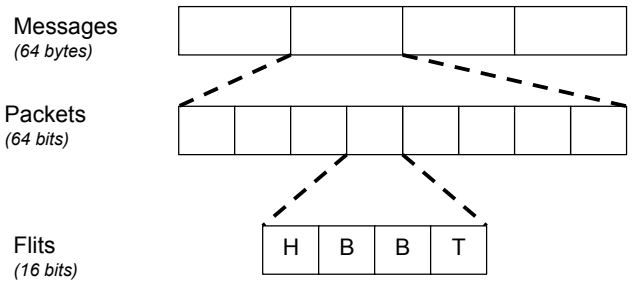


Figure 2.2: Breakdown of a 64-byte interconnect message into 8 packets and 4 flits.

Similar to messages, packets can further be subdivided into *flits*. Flits are short for flow control digits and represent the unit of bandwidth and storage allocation in a network. Unlike messages or packets, all flits do not contain destination or data information. Flits are categorized as head flits which contains packet and route identifiers, body flits which contain payload data, or tail flits which indicate the end of a packets. If the packet size is the same as the flit size, then one flit can serve as both a head and tail flit simultaneously.

Finally, flits are segmented into *phits*, a physical unit corresponding to the channel width. Flits are predominantly the same size as phits in on-chip networks. As such, the remainder of this thesis will refer to flits as the lowest communication unit within an interconnection network.

Figure 2.2 illustrates the units of communication within a system. In the example, a 64-byte message is first divided into 8, 64-bit packets. Since the channel size of the interconnect is 16-bits, the packet is then divided into 4 16-bit flits for data transfer. From the example, one can derive that the size of a flit is determined by the width of channel connecting two components.

2.1.2 Metrics

Latency (or *delay*) is known as that time it takes one component to receive a message from another component. In Figure 2.1, Core_0 would communicate with Memory_3 by generating a message X and sending that message through the interconnect. The time it

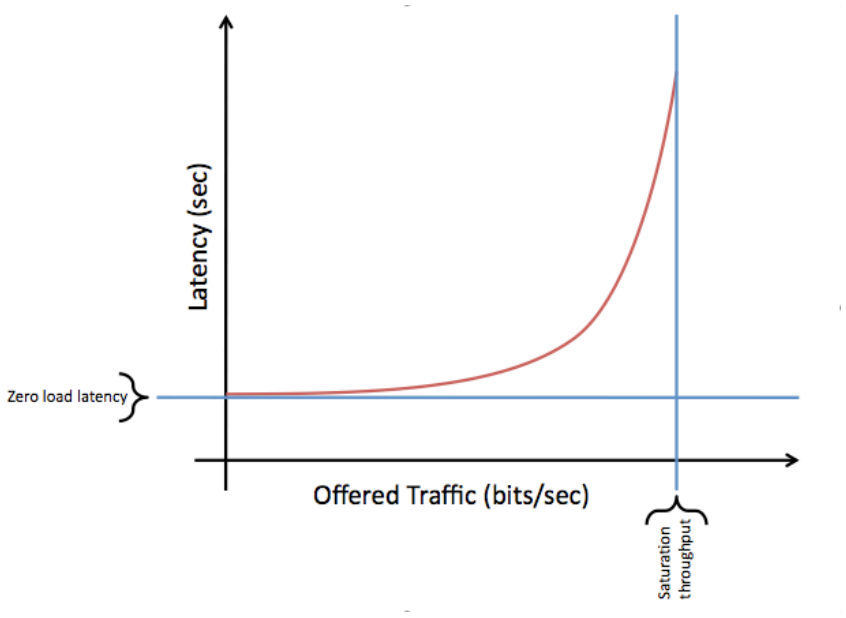


Figure 2.3: Offered Traffic v. Message Latency in a Interconnection Network.

takes for X to reach Memory_3 is then the communication latency for message X.

Additionally, Memory_3 would like respond to request message X with response message Y. The total time for message X and message Y to be completed is known as the *round-trip delay* for Core_0's request.

Typically, the latency of a particular message can be variable depending on factors such as contention for interconnect resources or the distance between the source and destination components. *Zero-load latency* refers to the optimal delay for any message—the case where a source component sends a message to it's destination without any contention.

Bandwidth is a throughput metric that measures how much data can be transferred over a interconnect during a given time period. Given an interconnect with a single channel, the bandwidth would then be the width of that channel multiplied by the frequency of the interconnect. *Bisection bandwidth* is a measure of how well an interconnect can communicate across the interconnect. If the interconnect is equally split into two parts, the bisection bandwidth is the bandwidth that would be achieved between those two parts. The *aggregate bandwidth* of the system is the sum of all the channels widths in the interconnect multiplied

by the same frequency as before. Aggregate bandwidth of the interconnect represents the maximum achievable throughput that can be maintained by the interconnection network.

The amount of traffic that one or more components send to the interconnection network is known as the *offered bandwidth*. Figure 2.3 shows how a interconnection network may be analyzed for a given amount offered bandwidth from one of it's components. As one would expect, the amount of aggregate bandwidth (throughput) that the system can sustain will eventually saturate and provide the *saturation throughput* for the interconnect under evaluation. Thus, the zero-load latency and saturation throughput lines can be viewed as the limiting factors for any component's offered traffic.

2.1.3 Arbitration

In cases where more then one component is using an interconnect channel as a shared resource, an arbitration scheme must be implemented to resolve conflicts. From Figure 2.1, it is apparent that all components have their own channel to the interconnect but the interconnect fabric itself is shared amongst components. If one were to imagine that interconnect fabric as a single channel that could only service one component at a time (*i.e.* a bus), then a conflict scenario would arise when multiple memory or core components would want simultaneous access.

This conflict resolution can be aimed toward providing fairness amongst contending components, providing priority for important components, or guaranteeing metrics such as bandwidth or latency for a subset of components. The requirements of a particular interconnect to arbitrate under specific constraints can then be labeled as that system's *Quality-of-Service (QoS)* policy.

Priority arbiters resolve shared resource contention by granting the component with the highest priority value access to interconnect. The most straightforward QoS policy is "static priority" in which fixed priority values are assigned to components. While static priority is a easy to implement QoS scheme, they are vulnerable to starvation of low priority compo-

nents. Dynamic priority schemes can change priorities according to system state. Lahiri *et al*. [69, 70] proposes a communication architecture tuner (CAT) that monitors the system, predicts the importance of future transactions, and alters the priorities of the underlying interconnect arbitration scheme. Interconnects like Satpathy *et al*.'s XRAM [99] provide a programmable priority substrate that can be dynamically tuned at the system's request.

Other architectures attempt to implement starvation free, fair QoS policies. These schemes aim to equally distribute the available interconnect bandwidth amongst components. Round-Robin (RR) schemes are perhaps the most straightforward and grant priority in a circular fashion to requesting components. A drawback of RR priority is that the maximum wait to get top priority from a RR arbiter rises as the number of connected component increases. In systems where there are messages that are more critical then others, this could create performance degradation. McKeown's iSlip algorithm [79] optimizes RR priority by de-prioritizing the Most-Recently Granted (MRG) component on the RR priority list. Implementations of the Least-Recently Granted (LRG) schemes [25, 100] similarly provide fairness through MRG demotion and can guarantee that all requesting components that haven't had access to the interconnect will be granted access before any components that have been previously granted.

Time Division Multiple Access (TDMA) arbitration schemes [88] are part of a class of arbiters that attempt to guarantee metrics such as bandwidth and latency amongst conflicting resources. In particular, TDMA allocates each input on the bus a fixed time slot for use of the interconnect. Such a method guarantees bandwidth and latency for a components on a bus. If a component does not want to use the bus during it's fixed time slot, RR priority can be used to pick between other available components. This type of hierarchical TDMA/RR scheme is featured in the Sonics SMART Interconnect [108]. Lahiri's LOTTERYBUS [67, 68] arbiter guarantees bandwidth by assigning tickets to components on the bus and using a 'lottery manager' circuit to replicate a lottery (i.e. random drawing) for arbitration cycle winners. Variations of LOTTERYBUS use the arbitration history [125]

or weights on the contending components [19] to direct the lottery manager and possibly enforce real-time bandwidth guarantees [73].

The final class of arbiters discussed in this section is called queueing arbiters [25]. These mechanisms enforce a FIFO priority on time stamping incoming interconnect requests. The implementation of such a queuing policy can be complex as tree-logic may be needed to find the oldest request and care must be taken to handle the overflow of timestamps. If those issues can be taken care of, a queueing arbiter would provide the strongest fairness possible out of all the QoS arbitration options.

2.1.4 Limitations

At any particular technology node, physical limitations such as the length of wires, the number of channels and channel width affect the frequency that a interconnect is run at and the energy that it consumes. For monolithic interconnects such as busses, the longest wire that needs to be communicated across is considered to be the *critical path* and consequently is a frequency limiter. Interconnects that have a high-radix (number of channels) or whose channel size is large can consume a high amount of energy during operation. Consequently, narrow channel [102], hierarchical [28], and multiple-clock domain interconnects have all been proposed as viable solutions for connecting components.

The following sections elaborates further on 3 common interconnect designs: busses, crossbars, and network-on-chip. Each have overlapping characteristics and tradeoff features such as simplicity, bandwidth, frequency, and QoS.

2.2 Busses

Busses are comprised of a set of shared wires amongst communicating components. In the simplest case, a bus can be viewed as a single, shared channel. Figure 2.4 illustrates a bus connecting cores to the memory.

To access the bus, a component must first request access to the bus from the bus arbiter.

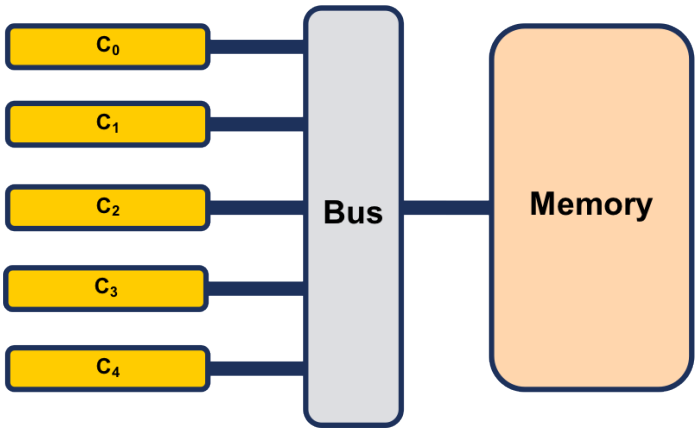


Figure 2.4: A bus interconnect connecting cores to memory.

The bus arbiter is typically a centralized arbitration unit in which all components can make their requests to. The arbiter provides contention resolution and priority to competing requests and will eventually grant a requesting component access to the bus.

Once access is granted, a component will broadcast its message to all components on the bus. This broadcast property of busses provides system-level benefits with regards to cache coherency and memory consistency. Since all requests have a single ordering point (the centralized arbitration), it's easier to maintain and enforce memory consistency in a bus-based system relative to systems that do not broadcast all accesses. Additionally, the visibility of requests to all components simplifies cache coherency protocols and allows components to update their cache state or respond to cache requests immediately after seeing a relevant request across the bus.

The frequency of a particular bus is dependent upon the length of wire needed to connect all the communicating components on a bus. The constraint of connecting all components on a singled shared bus line has typically limited busses from running at high frequencies (GHz range) as the number of cores in a system increases.

The zero-load latency of a flit to travel across a bus is again governed by the frequency of that bus. However, zero-load latency may not be achieved often on systems with a large number of components because of the sharing of the bus. Traffic due to memory requests,

coherence updates, and I/O demands can significantly increase the amount of contention for a shared bus. Since the bandwidth of a single bus is limited to the sequential fulfillment of only one request at a time, busses reach their saturation throughput faster then other, more distributed interconnects.

Modern bus-based systems combat these frequency and bandwidth issues by implementing multiple busses according to component types, message class, and other system characteristics. For example, ARM's AMBA bus protocol [5] specifies a high-performance bus (AHB) and peripheral bus (APB) to separate the latency and bandwidth requirements of performance critical components (*e.g.*, cores and memories) from less latency critical components (*e.g.*, timers and UARTs). The AXI and ACE extensions to AMBA also specify separate busses for control, data request, data response, and even cache coherency. Additionally, bus protocols also extract parallelism from the interconnect by allowing components to send bursty traffic (*i.e.*, multiple packets in sequence), pipeline transactions, and complete requests out-of-order [88].

Although split-transaction [35] and hierarchical [101] bus architectures have led to increased bus performance, the implementation of high-performance busses typically does not constitute a power-efficient interconnect. The overheads of wiring complexity and energy become prohibitive as the demand to drive arbitration and data lines at high frequencies increases as more components are added to the bus. As such, power issues as well as the previously mentioned contention issues threaten the future scalability of busses beyond 8-16 cores [25].

2.3 Crossbars

Crossbars, often referred to as just "Switches", are similar to busses in that there is a central connection between communicating components. Also similar to a bus, components must be granted access to the crossbar before they are allowed to drive their data across the interconnect. The key difference is that all components connected to a crossbar are

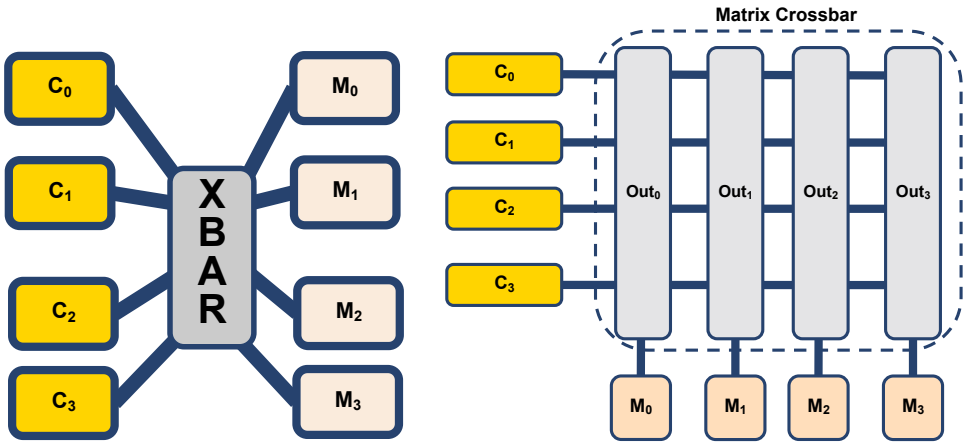


Figure 2.5: High-level view of a (a) 8x8 crossbar interconnect and an (b) 4x4 matrix crossbar connecting cores to memory. A separate crossbar connecting memory to cores would be needed to complete the circuit in (a).

available as separate channels to other components. Each channel contributes to the *radix* of the crossbar, such that a crossbar with 128 ports (*i.e.*,. 1 per separate component channel) is referred to as a radix-128 crossbar. As long as the crossbar has no requests with the same destination channel, messages on a crossbar can be processed in parallel. Using Figure 2.5b as an example, Core_0 can send a message to Memory_0 on the same cycle as Core_1 sends a message to Memory_1. Consequently, the crossbar can maintain the the same zero-load latency that a bus achieves despite having multiple components request access to the crossbar on the same cycle. Additionally, the independent output channels of a crossbar allow for higher bisection and aggregate bandwidth of crossbar interconnects relative to busses of the same channel size.

The increase in input and output ports on a crossbar can cause significant arbitration overheads as well as affect the critical path of the crossbar. Passas [90] implemented a radix-128 crossbar and found that the arbitration/scheduling area for his design was 60% of the total crossbar area. Thus, the frequency of a crossbar is degraded as the radix increases and the logic to needed to arbitrate amongst increasing components becomes more and more complex.

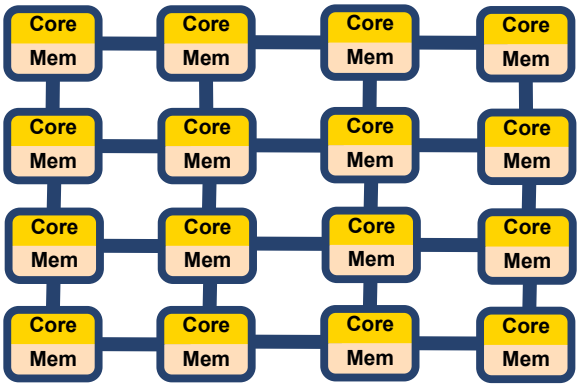


Figure 2.6: A Mesh Network-on-Chip Topology.

Since interconnect power is also adversely affected as the crossbar radix increases, many researchers have published that crossbars are simply infeasible for systems with a large number of components that need to be connected [66]. Although wiring complexity, area, and power are all concerns for the feasibility of high-radix crossbar, commercial chips like the Niagara2 [54] and IBM BlueGene/Q [43] use a crossbar for 16 and 18 cores respectively.

This work shows how a optimized crossbar using integrated arbitration and a a SRAM-like layout can scale to high-radices and support many-core systems. Chapter III will elaborate further on challenges and solutions to the high-radix switch problem. Chapter IV will present a system-level design for a high-radix crossbar system.

2.4 Network-on-Chip

The bandwidth issues in busses combined with the prohibitive implementation overheads of crossbars has led many architects toward Network-on-Chip interconnects when searching for a scalable on-chip communication fabric. While busses and crossbars centralize their communication logic, NoCs distribute the communication between components through the use of intermediate routing elements. Figure 2.6(a) shows an example of a NoC with Mesh topology. Each of the tiled components on the Mesh (core+memory) are

connected to it's own router. Within each router is a crossbar-like switch, which connects to it's home tile and at most 4 other routers. However, since the radix of these switches is small, the Mesh avoids the wiring complexity and power overheads that are seen in a monolithic crossbar system while maintaining high aggregate bandwidth throughout the interconnect. Because of these advantages, the tiled Mesh topology has been implemented in both the Intel TeraFlops [45] and the Tilera Tile64 [120] processors. Rings [40, 103] and spidergon [22] interconnects have also been used in commercial NoC chips.

While the distributed nature of a NoC enhances scalability, it also brings the issue of non-uniform communication amongst components. In cases where each NoC component has a slice of shared memory, this Non-Uniform Memory-Access (NUMA) can cause problems when the communicating components are far away from each other. In Figure 2.6(a), consider the situation in which the uppermost-left tile (A) requests data located in the bottom-right tile (B). No matter what path the requests take, there will be at minimum 6 routers that the request needs to traverse through to reach it's destination. Consequently, issues like quality-of-service enforcement can become significant as NoC systems continue to increase in number of components. Recent work has proposed solutions to these problems that map thread and data in a NoC aware fashion [42] as well as provide guarantees on QoS fairness [39, 71].

Since the ideal interconnect can vary greatly even within application domains, parameters such as topology and routing are of great importance when deciding what type of NoC to implement. The following subsections examine these details closer while acknowledging that an exhaustive look at all relevant NoC parameters is beyond the scope of this chapter.

2.4.1 Topologies

The choice of NoC topology has a direct effect on the performance and power tradeoffs of the system. In particular, the desired number of nodes (*i.e.*, components), zero-load

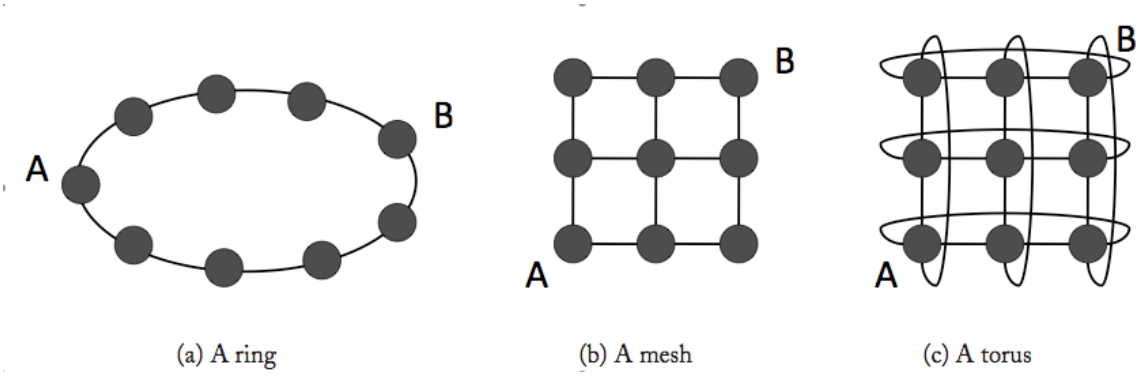


Figure 2.7: Common on-chip network topologies. Illustration from [52].

latency, and bandwidth are first-order constraints when building a NoC topology.

Systems with a small number of nodes may opt for a fully connected interconnect like a bus or crossbar (note: busses and crossbars can be viewed as a subset of the NoC domain), while systems with a large number of nodes tend to invite a more distributed NoC topology. Similarly, the required zero-load latency and bandwidth to and from arbitrary nodes provides a constraint on the the distance between any two nodes as well as the width of the NoC's channels.

The aforementioned design constraints manifest themselves into topology-specific characteristics such as degree, hop count, path diversity, and maximum channel load. The degree of a topology refers to the number of channels that a particular routing element connects. For example, the ring in Figure 2.7 has routing nodes of degree 2 and the torus has routing nodes of degree 4.

The hop count is the number of routing elements a flit must traverse in order to get from source to destination node. Zero-load latency is directly effected by hop count as even with no contention on the NoC, a message still must spend the required cycles to route from one node to the next. Maximum hop count and average hop count are typical metrics used when evaluating a topology. Computing the maximum hop count is typically straightforward: it is the largest manhattan distance from source to destination node. The average hop count is

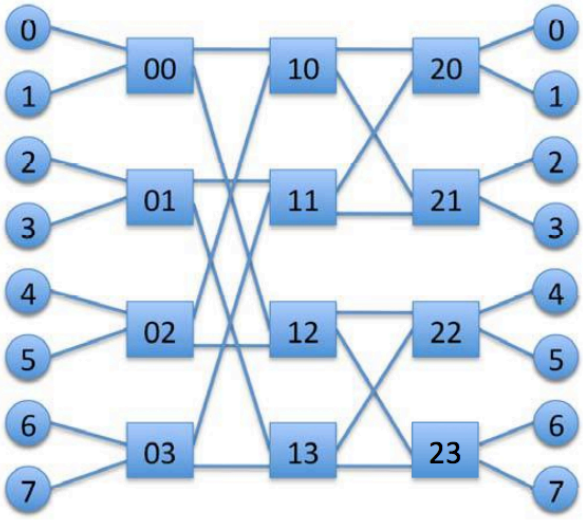


Figure 2.8: A 2-ary, 3-fly butterfly network. Illustration from [52].

defined as the average minimum hop count for all possible source to destination node pars in the NoC.

Path diversity refers to the number of shortest paths (i.e. minimum hop counts) that is available between any two nodes in the systems. Path diversity typically allows for greater routing flexibility as a system with high path diversity will have the ability to route around highly contended or faulty routing elements.

Finally, the maximum channel load represents the highest amount of offered traffic that a particular channel can accept before it reaches it's saturation throughput. For any channel in the system, an analysis similar to the one showed in Figure 2.3 can be done to identify the maximum channel load for the interconnect.

Once all design constraints are known, a NoC architect can select a direct or indirect topology to satisfy network requirements. Direct topologies are networks in which each routing element connects to a destination (or terminal) node whereas indirect topologies have an intermediate routing elements that connect to other routing elements but no terminal nodes. The Mesh, Ring, and Torus topology from Figure 2.7 are examples of direct networks and Butterfly network in Figure 2.8 illustrates an indirect network topology.

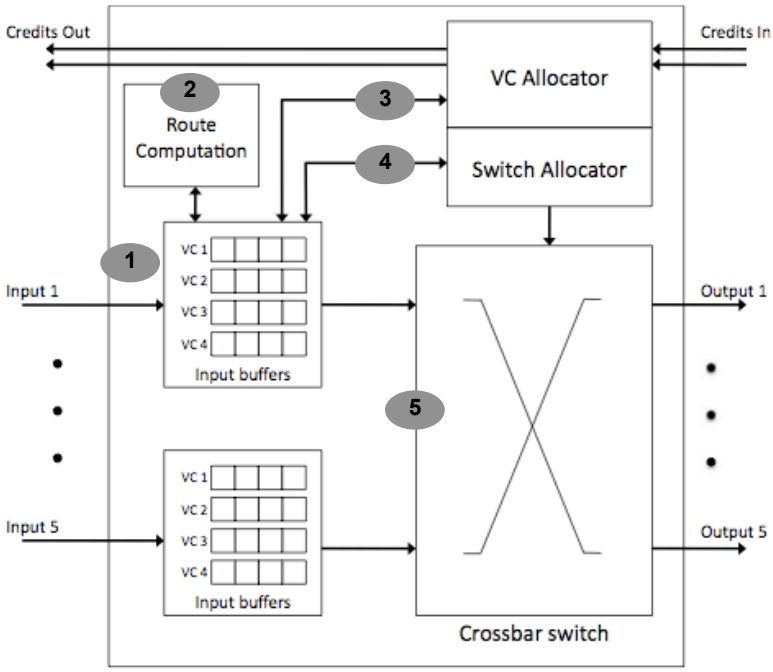


Figure 2.9: The Microarchitecture for a credit-based, virtual channel router found in a Mesh Topology (note: 5 inputs and 5 outputs). The circled numbers represent router pipeline stages.

2.4.2 Router Microarchitecture

Figure 2.9 illustrates the contents of typical NoC routing element. From a high-level, each router accepts an input flit, implements a routing algorithm, and uses the crossbar switch to send a flit to it's the next output port along it's routing path. Ideally, NoC routing would minimize the hop count between source and destination node in an area and energy-efficient manner. The example in Figure 2.9 also notes the 5 logical stages in NoC router: Buffer Write (BW), Route Computation (RC), Virtual Channel Allocate (VA), Switch Allocate (ST), and Switch Traversal (ST).

In the BW stage, an incoming flit is written into buffer space reserved for that input. That space can be in the form of a single queue for all incoming flits or multiple queues known as virtual channels (VCs). Providing virtual channels per input port assists the router in deadlock avoidance [26], Head-of-Line (HoL) blocking prevention [25], and in

providing dynamic routing algorithms [30].

Additionally, the BW stage provides the first level of flow-control to a NoC router. Flow control mechanisms determine how much buffer space is reserved per router and when messages can leave the router. The tracking of credits in Figure 2.9 supports flow control by allowing upstream routers to process information (*e.g.*, buffer space) from downstream routers. Message-based (or circuit-switched) flow control is bufferless as the necessary routing resources (links and router switch) are reserved before the message is sent from the source nodes. Store-and-Forward flow control [25] reserves enough buffer space for a packet in the router and will only send that packet to the next node (or router) once all the flits for that packet are first stored in the current router. Virtual-Cut Through [56] optimizes Store-and-Forward flow control by allowing the packet to start sending to the next routing node as long as that next routing node has sufficient resources to buffer the complete packet. Wormhole [27] flow control becomes more useful as packet-sizes become large and the requirement of a reserving a full-packet at a time decreases network utilization. Wormhole routing will transmit a flit to the next node as long as there is buffering for that single flit in the downstream router.

The output port for an incoming packet is determined in the RC stage. Most of the routing algorithms outlined in the Section 2.4.3 use a table-based routing scheme where a packet can find it's output port by using it's destination node as a index. Table-based schemes also are easily updated for use in dynamic (*i.e.*, adaptive) routing algorithms. Route computation can also be computed using a combinational circuit at each routing node or calculating the complete route at the initial sending node (source routing).

The VA stage starts the arbitration process for the router by choosing a ready Virtual Channel (VC) from each input port. Once a VC is selected, a flit from that input port's VC can vie for access to the crossbar switch.

Arbitration for the crossbar switch is done in the SA stage. Flits from each input port request access to the switch and a grant signal is sent back to indicate that a particular input

port can now send it's data through the crossbar.

The ST stage represents the cycles taken for a flit to pass through the crossbar and onto the appropriate output port. Once a flit reaches it's output port in the router it is then free to be traverse the link to either it's destination node or the next downstream router (note: this process is also be referred to as the Link Traversal (LT) stage in some literature).

Router pipeline optimizations include eliminating the routing stage via lookahead routing [34], bypassing buffer writing if there are no flits ahead in the input VC, and processing the VA and SA stages in parallel [91, 84, 85]. Since these optimizations can significantly increase the complexity of the router, their implementation is often dependent on target frequency of the network and whether the pipeline can satisfy critical path constraints. Additionally, performance from speculative optimizations is limited by the amount of offered traffic and contention at a particular router node.

2.4.3 Routing

Routing algorithms can broadly be classified as either deterministic, oblivious, or adaptive. Deterministic algorithms always traverse the same path from source to destination node. A commonly used deterministic algorithm is a dimension-ordered routing (DOR) algorithm. DOR algorithms are both simple to implement and deadlock free. For example, the X-Y DOR algorithm will always travel in the x direction first and then the y dimension. The Tilera Tile64 is an example of a system that uses DOR routing.

Oblivious algorithms select a path without regard to any dynamic network conditions (*e.g.*, congested or faulty routers). The implementation of these algorithms is often low-complexity (no network information is needed) and can often implicitly load-balance the network. A routing algorithm that randomly selects it's path would be considered oblivious as well as load-balancing. Valiant's algorithm [116] randomly selects a intermediate node between source and destination to achieve this effect. The average hop count between any two nodes would be increased in such routing methods as the shortest path between

two nodes is usually disregarded in favor of the random destination node. Nesson [86] optimizes Valiant's by restrict the randomization to minimal paths only. This "Minimal Valiant's" sees the dual benefits of load-balanced traffic and reduced average hop-count.

Adaptive routing algorithms can choose a message's path dynamically. For instance, if the routing algorithm detects traffic along a particular path it can route around the congested nodes [106, 24]. The downside of adaptive routing algorithms is the complexity of implementation. Resources (*e.g.*, logic and buffers) must be maintained to monitor network conditions and the adaptive routing algorithm also must ensure deadlock-free operation. Typically, turn model routing [37, 21] is used to avoid this deadlock. These schemes specify a ordering of legal turns along any message's path that will prevent cyclic dependency and in turn prevent deadlock. Alternatively, there has been work by Duato [31] that allows for full-routing adaptivity but ensures deadlock free operation through flow-control mechanisms.

2.5 Related Works

The paradigm shift toward many-core systems has led to a renewed interest in interconnect research and a transition from traditional bus-based systems [66] to more sophisticated topologies, including hierarchical bus models [11, 28], rings [1, 117, 40, 103], spidergon networks [13], mesh network-on-chips [7, 120], flattened butterfly on-chip networks [61], express cube on-chip networks [38] and crossbars [54, 3, 78, 127]. The ability to provide uniform cache access latency makes crossbars an appealing option because predictable cache access latencies allow for quality-of-service guarantees and ease of programming. In addition, previous research has shown that crossbars can enable performance benefits in coherence protocols [20] as well as the construction of cache hierarchies [72]. While some studies have noted that link latency can increase to a point that it would be intolerable compared to an NoC system [14, 117], this thesis proves through detailed floorplans and spice analysis that this is not always the case.

The *Swizzle-Switch Network* proposed in this thesis optimizes the crossbar interconnect allowing for high performance many-core systems with minimal power and area overheads readily scaling to support 64-core systems. This work demonstrates the benefits of a crossbar-based architecture for systems that are required to support a wide range of communication patterns.

Additionally, this thesis evaluates the future scalability of crossbar and bus architectures in the presence of 3D-stacking technology. There have been many works that research 3D technology for use in logic circuits [92], memory optimizations [113, 75] and full-system architectures [57]. However, this work focuses on the use of 3D integration technology for scalable, high-radix interconnects.

Related works have also leveraged the benefits of high-radix switches by using narrow channels to increase crossbar radix and build larger systems from these high-radix building blocks [62] [102]. Additionally, there has been similar work analyzing crossbar interconnects for large-scale CMPs. Some assume idealized crossbars with minimal latencies to calculate best-case performance [96], others use crossbars to connect small clusters of cores in a hierarchical system [118], and yet others study pipelined/buffered crossbar systems [89, 64]. The *Swizzle-Switch Network* differs from these systems because it uses a flat, non-buffered interconnect based on detailed floor-planning, SPICE simulation, and measured silicon results of the *Swizzle-Switch* crossbar.

CHAPTER III

Swizzle-Switch: A High-Radix, Self-Arbitrating Crossbar

Chapters I and II motivate the performance-scaling needs of many-core systems as well as discuss the inability of bus-based interconnects to support scaling from multi- to many-core. This thesis makes the case that high-radix interconnects are needed to support future many-core architectures and in Chapter IV designs a crossbar-connected 64-core system, the *Swizzle-Switch Network*, to prove that such a interconnect is both feasible and advantageous to traditional Network-On-Chips.

This chapter focuses on the *Swizzle-Switch*, an essential building block of the *Swizzle-Switch Network* (SSN). The SSN is a high-radix crossbar system built using a number of *Swizzle-Switch* components. Each *Swizzle-Switch* component employs several techniques to reduce the area and power overhead of high-radix crossbars. After this chapter explains the theory of operation for the *Swizzle-Switch*, it is then scaled to 32nm using measured crossbar power and performance from 65nm and 45nm test chips. The research conducted was done in collaboration with Sudhir Satpathy, Nathaniel Pinckney, Ronald Dreslinski, and Reetuparna Das and portions of this chapter are published in [99, 98, 100, 97].

After detailing the *Swizzle-Switch* crossbar circuit, the key architectural challenges of building a MOESI-coherence, 64-core chip multiprocessor using *Swizzle-Switches* are further described in Chapter IV.

3.1 Overview

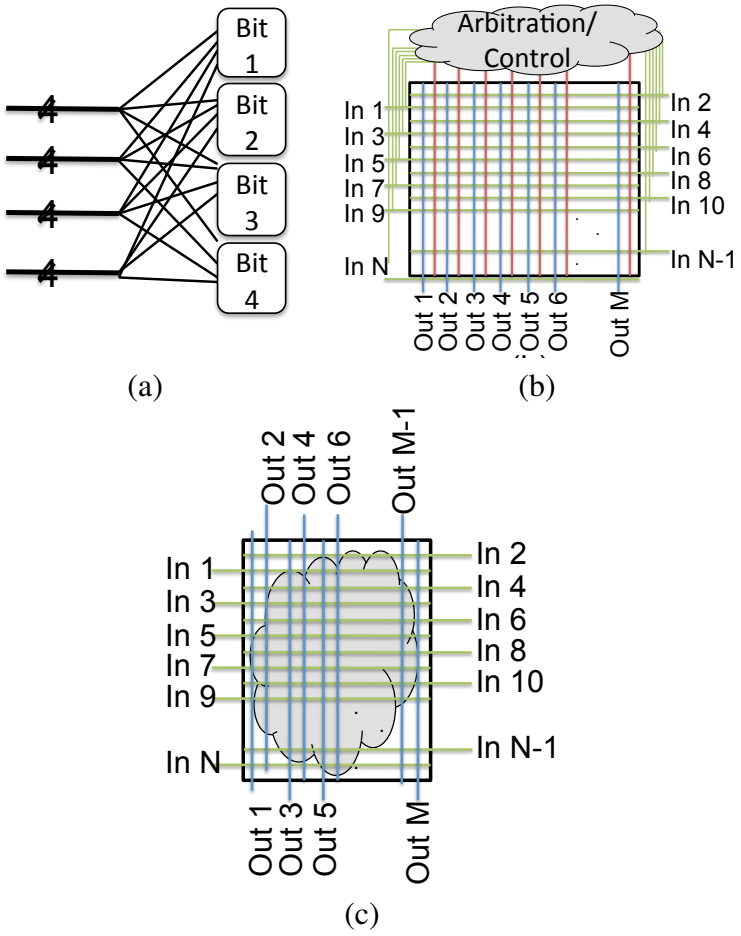


Figure 3.1: High level view of (a) bus interleaving required for mux-based crossbars; (b) a traditional matrix style crossbar with arbiter/controller consuming space and requiring additional input wires; (c) the proposed *Swizzle-Switch* design that reuses input/output busses for programming/arbitration of the crossbar with the arbitration logic placed under the dense metal interconnect.

Conventional mux-based crossbars suffer from a layout challenge at high bus widths because of complex wire interleaving within the crossbar itself. Figure 3.1(a) shows how four buses each with four bits must be interleaved to connect to a mux-based crossbar. To avoid these interleaving structures, more recent crossbars use matrix-style structures.

Figure 3.1(b) shows a typical matrix-style crossbar, where the connection to each output

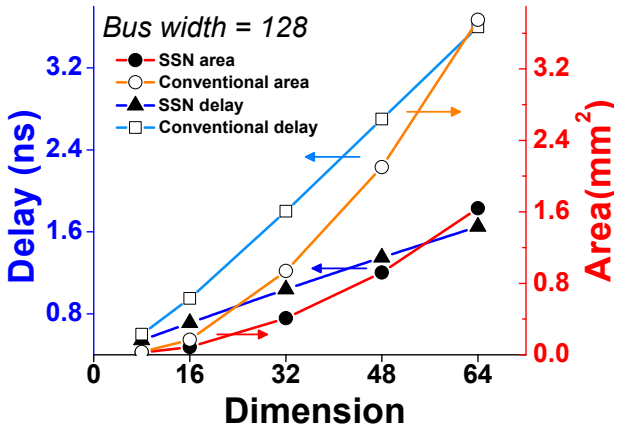


Figure 3.2: Scaling trends of the *Swizzle-Switch* Interconnect in 32nm vs. a conventional crossbar (Simulated).

is made at a crosspoint inside the crossbar, the inputs can be in any order and no interleaving of bits from buses is required. Conventionally, these matrix-style interconnects consist of a *crossbar that routes data* and a *separate arbiter that configures the crossbar*. This decoupled approach poses two hurdles to scalability: (1) the routing to and from the arbiter becomes more challenging as the number of sources and destinations increase and (2) the arbitration logic grows more complex as the radix of the crossbar increases. Consequently, the conventional matrix-style crossbar suffers from poor scalability in delay, area and power when scaled to large core counts.

To overcome these limitations, the proposed *Swizzle-Switch* crossbar can be used to replace conventional matrix-style crossbars. Although the *Swizzle-Switch* has the same asymptotic behavior as the matrix-style crossbar, $O(n^2)$, it uses circuit techniques to reduce the multiplicative constants of that behavior to readily scale to at least 64 cores. The *Swizzle-Switch* combines the routing-dominated crossbar and logic-dominated arbiter by *embedding the arbitration logic within the router crosspoints*. Furthermore, it *reuses input/output buses for arbitration*, producing a compact design. Figure 3.1(c) shows a high-level *Swizzle-Switch* design. To reduce power, the *Swizzle-Switch* uses SRAM-like technol-

ogy with low-swing output wires and a single-ended thyristor-based sense amplifier [99]. To understand the scalability of the *Swizzle-Switch* interconnect compared to a conventional crossbar design, a series of layouts across a wide range of radices is generated. Figure 3.2 shows the results of this analysis. The *Swizzle-Switch* design scales far better to higher radix designs, consuming 2.7× less area and performing 2.6× faster at a radix of 64.

3.2 Layout and Data-Transmission Phase

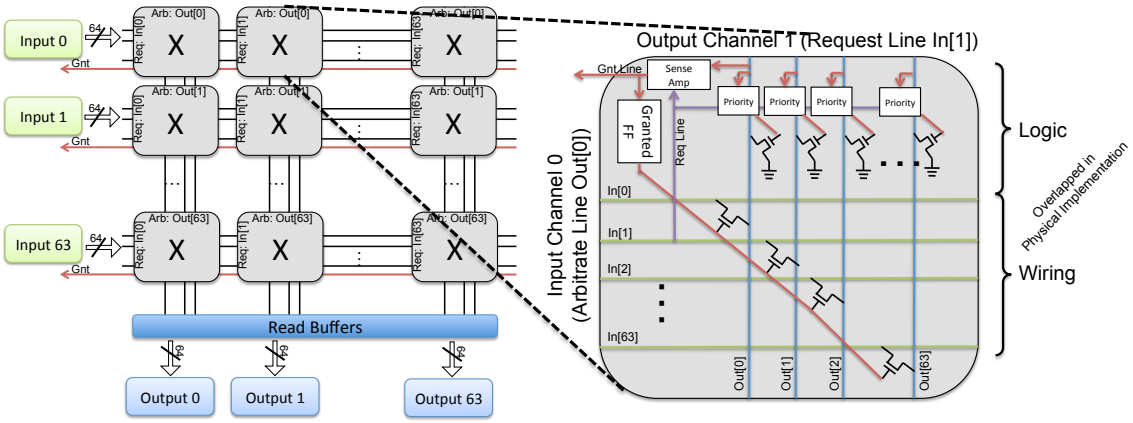


Figure 3.3: Circuit implementation of the *Swizzle-Switch* Interconnect. Each output *column* in the interconnect uses the same request bit from each input bus. Each input *row* uses the same bit from each output bus to perform arbitration. The expanded view of the crosspoint shows the stored configuration and crosspoint connections for each bit. It also shows the programming of priority bits using the output bus. Because the crosspoint is for Input Row 0, the arbitration sense amp is on *output wire 0*. Similarly, because it is Output column 1, the request line is drawn from *input wire 1*.

As shown in Figure 3.3, the *Swizzle-Switch*'s input and output buses run perpendicular, creating a matrix of crosspoints each containing a storage element to designate connectivity. Along a column (output), at most one connection can be made, allowing each output to connect to at most one input. Along a row (input), multiple connections can be made. This allows a single input to multicast to a subset of outputs, or broadcast to all outputs.

Every output channel operates **independently** in one of two modes, *data-transmission*

or *arbitration*. When an output channel is not allocated, it enters arbitration mode until an input is granted access to the output channel. Once the output channel has been granted, it transitions to data-transmission mode. Data transmission can continue for several cycles as the input channel transfers the complete payload. Once the input is finished transmitting data, it releases the channel and allows the output channel to move back into arbitration mode.

The circuit details of the data transmission phase are illustrated in Figure 3.3. During data transmission, the output buses are pre-charged to '1'. The input channel then drives the horizontal wires with the data. At crosspoints where the "Granted FF (Flip Flop)" stores a '1', the input bitlines are coupled to the pre-charged output bitlines with a pass gate. If the input bitline is '0', the output bitline will discharge and the sense amplifier at the read-buffer will sense the data. The "Granted FF" uses a thyristor-based sense amplifier to set the enabled latch, which only enables the discharge of the output bus for a short period of time, reducing the voltage swing on the output wire. This reduced swing coupled with the single-ended sense amplifier helps to increase the speed, reduce the crosstalk, and reduce the power consumption of the *Swizzle-Switch*.

3.3 Arbitration Phase

Any output channel that is not in the data transmission phase is in arbitration phase. In this phase, each output channel will grant a single input channel access to transmit data. The input channel with highest priority is granted access. The *Swizzle-Switch* uses an *inhibit-based approach* to accomplish this arbitration. When an output channel is requested, input channels will inhibit other inputs with lower priority. Consequently, managing the priorities of each input allows for the implementation of various priority schemes. There are two novel aspects of the arbitration design: (1) The priority bits stored in each crosspoint are used to determine the winner of the arbitration. (2) Each input re-purposes a particular

bit of horizontal input bus to assert a request signal and is assigned a particular bit of the vertical output bus to use as an inhibit line.

3.3.1 Arbitration Mechanism

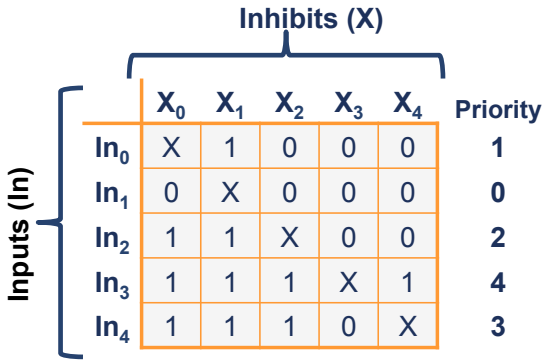


Figure 3.4: Conceptual example of *Swizzle-Switch* Arbitration. A matrix represents one complete output column. Each output column arbitrates and transmits data independently of other output columns.

The conceptual view of inhibit-based arbitration of a single output column for a 5-input *Swizzle-Switch* is shown in Figure 3.4. The arbitration for an output channel can represented by a matrix (M) of requests (In) and inhibits (X). A row of bits in M correspond to the storage elements labelled "priority bits" in the expanded crosspoint view in Figure 3.3. An input In_i inhibits an input In_j if and only if the entry $M_{(i,j)}$ is 1, indicating that In_i has priority over In_j. In this example, if input In_0 and In_1 are both arbitrating for the output channel then In_0 would inhibit In_1 (since $M_{(0,1)}$=1) and win the arbitration between the two requesting inputs. However, if In_0 and In_2 were in arbitration with each other, then In_2 would win the arbitration by inhibiting In_0 (since $M_{(2,0)}$=1). The priority for an input is then the number of *other* inputs that it can inhibit.

Figures 3.5 and 3.6 illustrates the operation of the arbitration circuit. The same priority scheme from Figure 3.4 is used. The vertical output $Out\ K_i$ bit-line is repurposed as inhibit line X_i during the arbitration phase. The input channel In_3 has 1's stored in all its priority

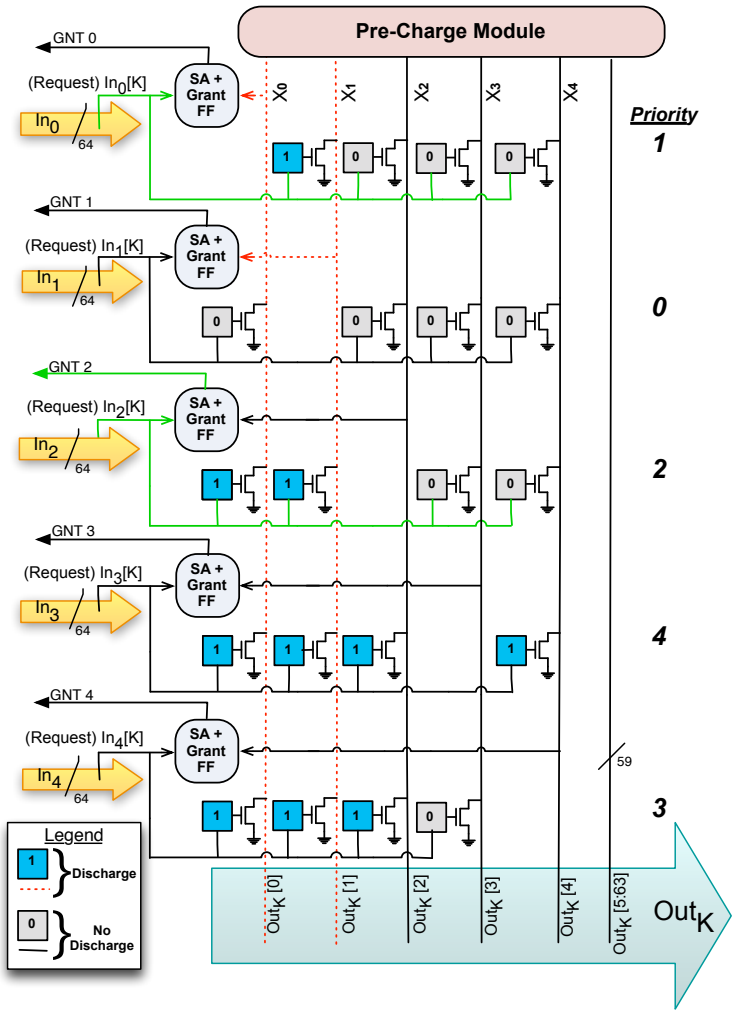


Figure 3.5: A 5x3 crossbar showing arbitration circuits. Each output column arbitrates and transmits independently.

bits $M_{3,j}$ and hence has the highest priority. The input channel In_2 has priority over only inputs In_0 and In_1 (because only these priority bits are set). At the start of arbitration, all inhibit lines are precharged. Then, for each competing input channel, if the priority bit (i.e. $M(_{i,j})$) is set, the corresponding inhibit line (i.e. X_j) is discharged via a pass transistor. Each input channel In_i monitors inhibit line X_i to determine if it won the arbitration. If the inhibit line is discharged, a higher-priority channel must have requested the output and consequently In_i loses the arbitration. Conversely, if the inhibit line remains pre-charged, then no higher-priority channel requested the output. The arbitration result is latched in the

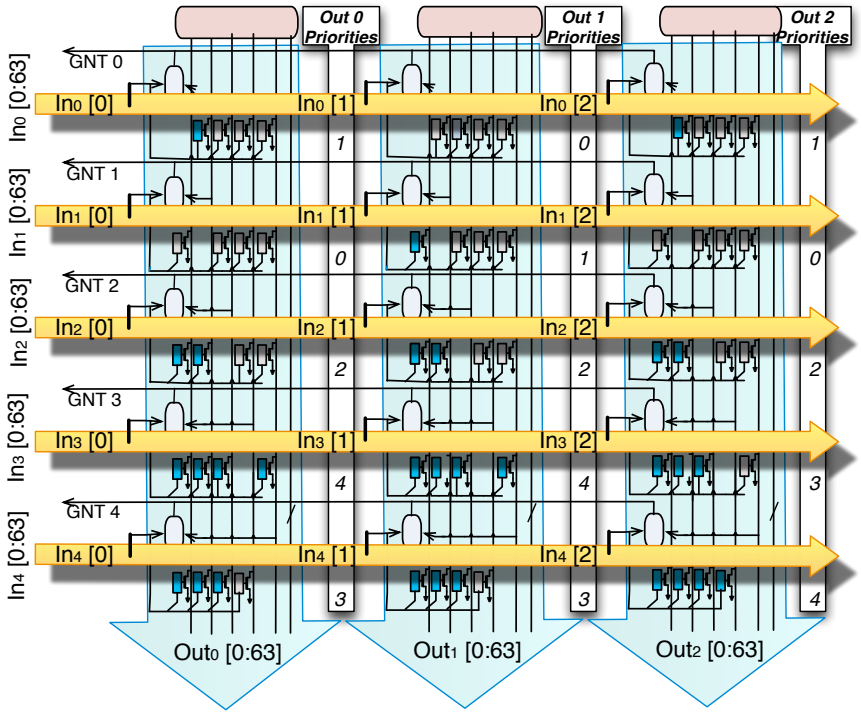


Figure 3.6: Detailed blowup of arbitration for the K^{th} output column of a 5-input *Swizzle-Switch* interconnect.

"GrantedFF" to set up the connection for data transmission.

Note that though these examples illustrate unicast requests, each input can request multiple output columns. Together, the bit-lines of an input port constitute a multi-hot signal to request a subset of output channels. The priority bits stored in the different crosspoints are used to determine the winner of the arbitration. By updating the priority every time the channel is granted, fair arbitration can be achieved.

3.3.2 Least Recently Granted

Fair scheduling algorithms can be implemented in the *Swizzle-Switch* by resetting and setting the appropriate inhibit bits in the arbitration matrix. Figure 3.4(b) shows how *least recently granted* (LRG) priority can be achieved using the *Swizzle-Switch*'s inhibit-based priority scheme. In this example, inputs 0, 2, and 4 are all arbitrating for the output channel.

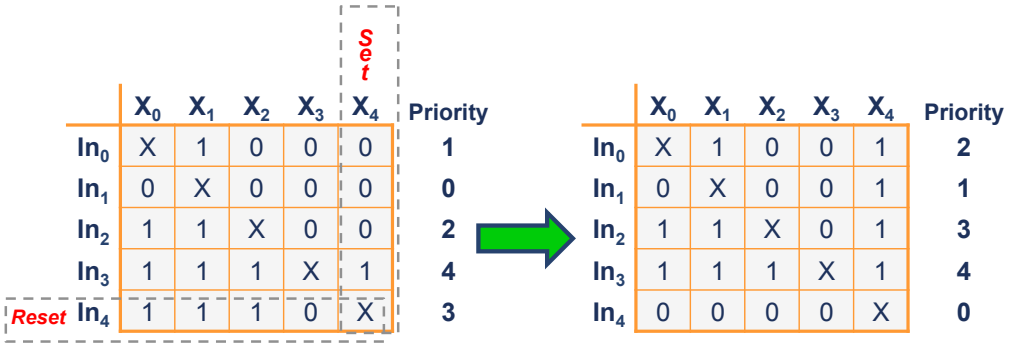


Figure 3.7: Least-Recently-Granted priority update of a *Swizzle-Switch* output.

Input 4 wins the arbitration since it has the highest priority amongst the arbitrating requests. To achieve an LRG update, all bits in the row of In_4 are reset to enforce that input 4 can not inhibit any other request in the matrix. Next, all bits in the inhibit column of X_4 are set. This enforces that all requests can inhibit input 4 during the next arbitration cycle. Thus, the combination of set and reset operations achieves LRG by giving least priority to input In_4 and incrementing the priority of all other inputs that previously had lower priority then In_4 by 1. LRG arbitration helps to ensure starvation-free operation.

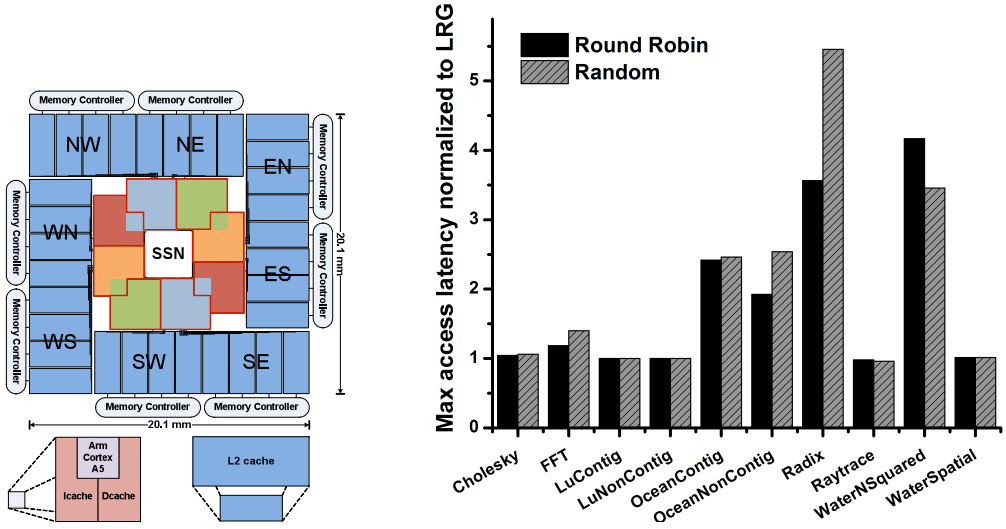


Figure 3.8: (a) 64-Core *Swizzle-Switch Network* System (b) Maximum Request Latency for Random and Round-Robin Arbitration Policies (Normalized to LRG).

The impact of LRG arbitration is further seen by comparing the algorithm to Random and Round-Robin schemes. Figure 3.8 charts the maximum latency of the arbitration algorithms. Each algorithm is run using the SPLASH 2 benchmarks [122] on a 64-core *Swizzle-Switch Network* system (detailed in Chapter IV). The results show that LRG arbitration reduces the worst-case request access latency by 1.83× and 2.03× on average over round robin and random arbitration schemes, respectively.

3.3.3 Most Recently Granted (MRG)

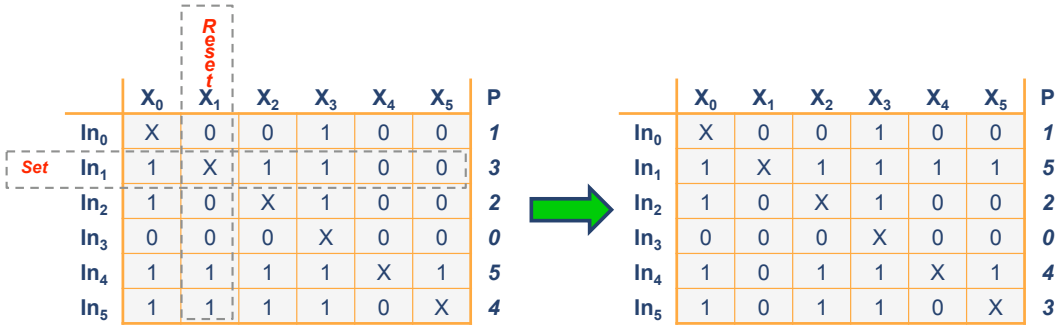


Figure 3.9: Most Recently Granted priority update of a *Swizzle-Switch* output.

Applications that desire greedy arbitration algorithms (e.g. prioritizing bursty traffic patterns) can use the *Swizzle-Switch*'s Most-Recently Granted (MRG) priority update. Figure 3.9 provides an example of MRG arbitration for an output on the *Swizzle-Switch* with 6-inputs. MRG based update is accomplished by setting all the priority bits along the In_1's row and resetting all the priority bits along the X_1 inhibit column. This set and reset operation provides In_1 with the highest priority and downgrades all priorities that were greater then In_1 by one.

3.3.4 Selective LRG and MRG

Selective LRG and MRG arbitration refers to the ability of the *Swizzle-Switch* to update priority on a specified subset of outputs. This type of arbitration can be useful in cases

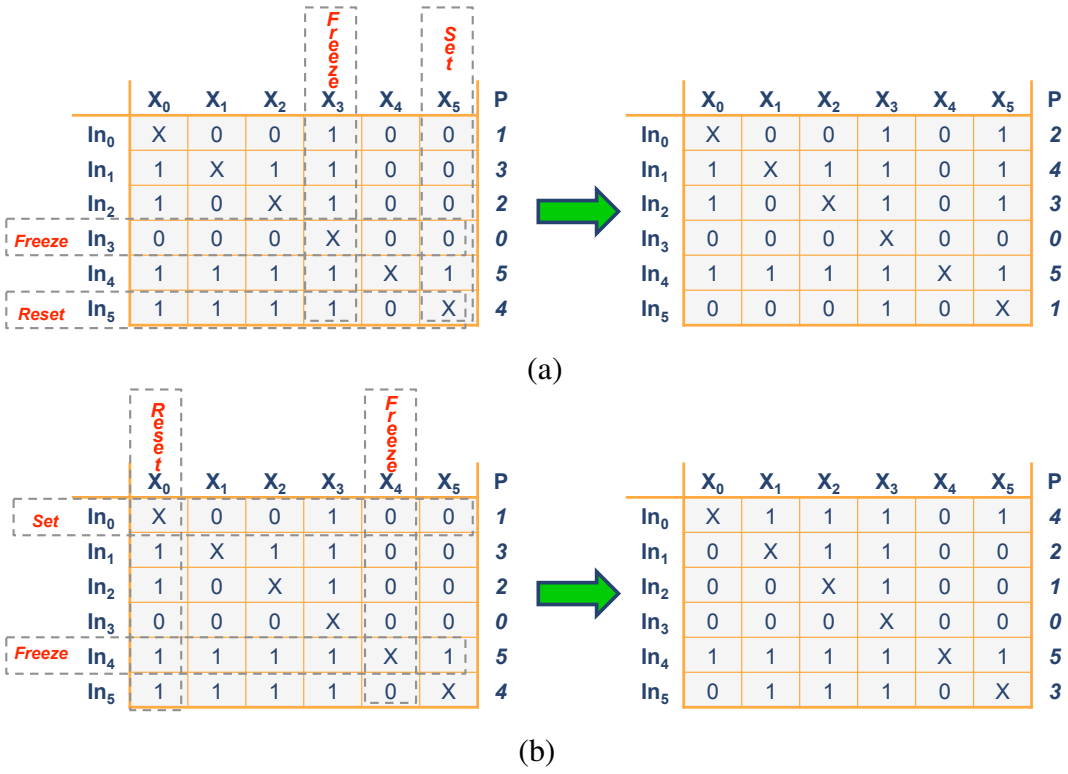


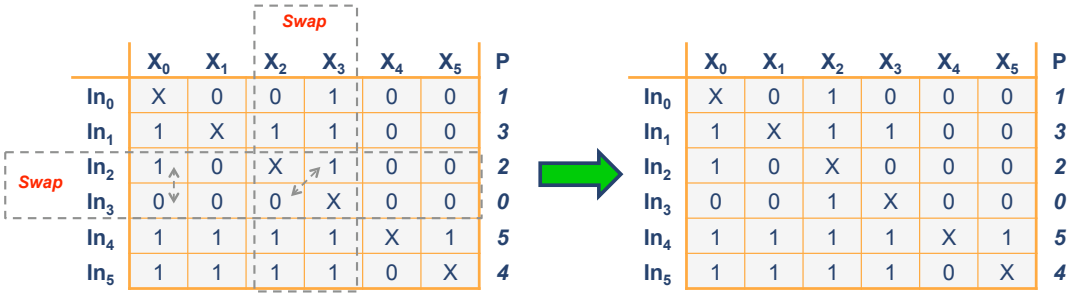
Figure 3.10: Selective LRG (a) and MRG (b) priority update of a *Swizzle-Switch* output.

where a subset of components must maintain a constant priority.

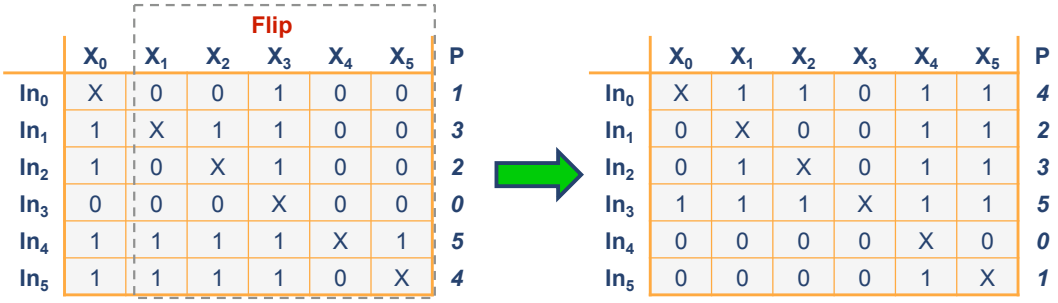
In the standard LRG scheme, the input that used the output most recently is downgraded to have the least priority while all inputs with lower priorities get upgraded one priority level. In the example shown in Figure 3.10(a), In_5 has a priority level of 4 and has used the channel most recently. However, the Selective LRG (S-LRG) arbitration in this example does not intend to downgrade In_5 to priority level 0, but to an intermediate priority such as 1. To accomplish this, S-LRG must identify certain rows and columns that need to be frozen. In this case, all columns corresponding to priority bits that are high in In_3 are frozen as shown in Figure 3.10(a). Simultaneously, all rows corresponding to priority bits that are low in the X_3 (which is the inhibit line for In_3) are also frozen. Lastly, the all the priority bits in the In_5 that aren't frozen are reset and those in the column for X_5 that aren't frozen are now set. These operations fulfill the S-LRG requirements. The update process for Selective MRG (S-MRG) is similar except that the set and reset operations for the most

recently granted input is updated the same way that is done in Section 3.3.3. Figure 3.10(b) shows an example of a S-MRG being performed where In_0 has just received access to the output but the priority for In_4 is maintained throughout the update process.

3.3.5 Priority Swap and Reversal



(a)



(b)

Figure 3.11: Priority Swap (a) and Reversal (b) of a *Swizzle-Switch* output.

Support for priority swap and priority reversal algorithms further allows the *Swizzle-Switch* to enable arbitrarily complex schemes amongst the its connected inputs. The priorities of 2 inputs can be swapped (without affecting priorities of other inputs) by swapping the priority bits in their corresponding rows and those in the columns corresponding to their priority lines as shown in Figure 3.11(a). In this example the priorities for In_3 and In_4 are swapped. In the physical realization of this technique, already existing word-lines will be used to swap priority bits between columns and bit-lines to swap priority bits between rows. In a single cycle, any two priorities can be swapped. The *Swizzle-Switch*'s unique priority encoding scheme also allows reversing the priority of all inputs instantaneously by flipping

all the priority bits as shown in Figure 3.11(b). Rather than flipping all the bits, a more energy-efficient physical implementation might use a multiplexer to invert the priority. An added benefit would also be the realization of this functionality without spending a clock cycle.

3.3.6 Round Robin

More traditional arbitration algorithms such as Round Robin (RR) can also be easily implemented in a *Swizzle-Switch*. Since a RR algorithm simply steps through the list of inputs and rotates priority, it can be abstracted onto either the LRG or MRG arbitration algorithms. Once the highest priority is established within the *Swizzle-Switch*, a RR update can be iteratively applying the LRG update to the highest priority on every arbitration cycle. Alternatively, a MRG update can be used on the lowest priority of arbitration cycle and achieve the same RR arbitration effect.

3.3.7 QoS Arbitration

In a 64× 64 *Swizzle-Switch* it might take a message 64 cycles to win arbitration in the worst case when all inputs collide. To assist critical messages in reaching their destination early, the *Swizzle-Switch* can also feature a 4- level message-based QoS arbitration technique that allows only input buses with the highest message priority to arbitrate for the channel as shown in 3.12. A 2-bit message priority is decoded into a 4-bit thermometer code at the crosspoint, which is used to selectively discharge priority bit-lines comprising the QoS priority bus. A multiplexer samples one of those priority bit-lines using its own message priority and the input bus progresses to the LRG arbitration cycle if the monitored priority bit is not discharged. Using separate wires for QoS arbitration incurs 3% area overhead. However, the additional QoS arbitration cycle can be overlapped with the prior routing operation for the output bus, avoiding a latency penalty.

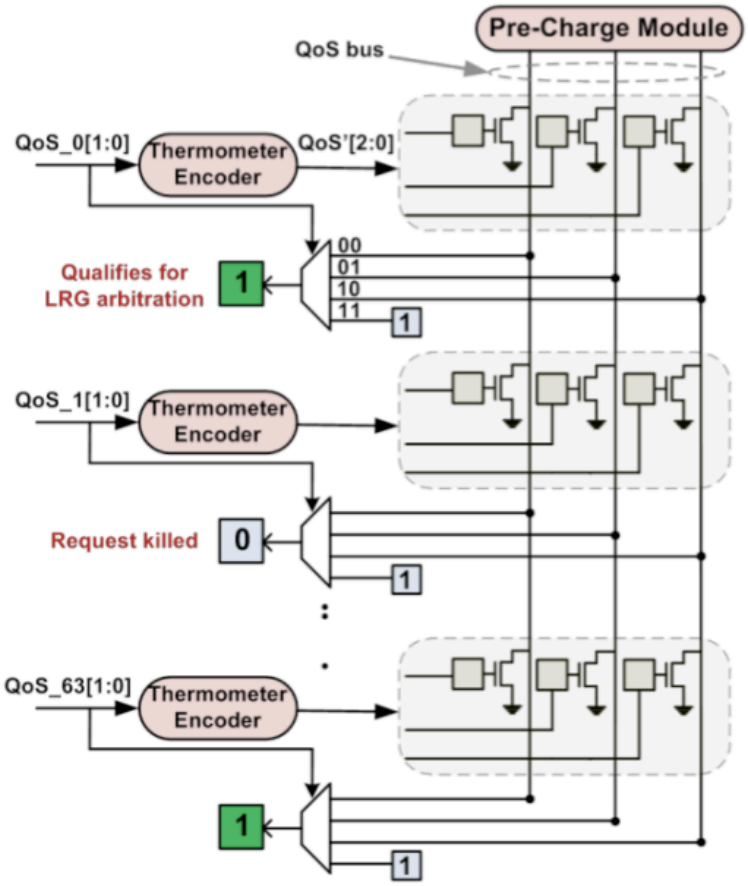


Figure 3.12: A Quality-of-Service Circuit for the *Swizzle-Switch*.

3.4 Silicon Validation

Previous work has shown the feasibility of the *Swizzle-Switch* building block with a fabricated and tested silicon prototype [100]. The prototype chip was manufactured in a commercial 45nm technology and consisted of a 64×64 Swizzle interconnect with 128-bit busses. The total size of the Swizzle interconnect was 4mm^2. The interconnect was driven by synthetic traffic generators (including support for broadcast and multicast traffic) with a built-in test circuit that verified correct transmission. Figures 3.13 and 3.14 shows the die photo of the silicon test chip. Measurements of the chip show that at full voltage the Swizzle interconnect operates around 559 MHz and provides 4.47 Tbps of bandwidth. The total power of the interconnect was 1.32W at full voltage under a 20% switching factor.

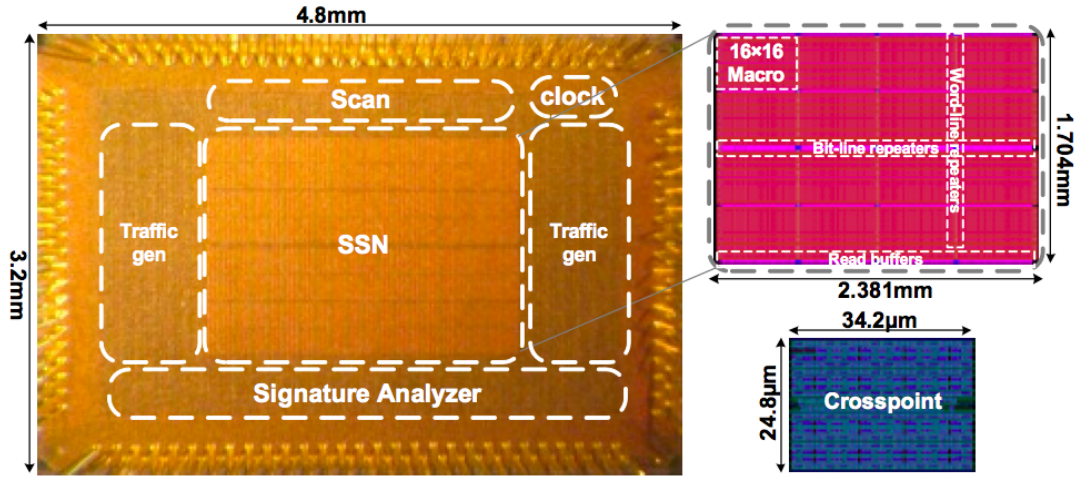


Figure 3.13: Die Photo of the 45nm silicon test chip.

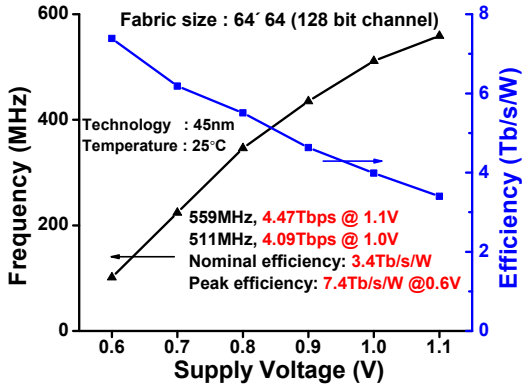


Figure 3.14: Measured Frequency and Bandwidth Efficiency of the silicon test chip from 3.13.

3.5 Enhanced 32nm Design and Analysis

SPICE modeling is used to scale the design of the *Swizzle-Switch* to 32nm. All results are validated against the 45nm test chip. In addition, the crossbar is further optimized for higher frequencies at high radices. The schematic-based SPICE crossbar model includes word-/bit-line drivers, parasitic loads, and worst-case coupling capacitance to neighboring wires. For a specific radix and bus width, driver sizes are optimized to maximize crossbar frequency and utilize the additional metallization layers to reduce area.

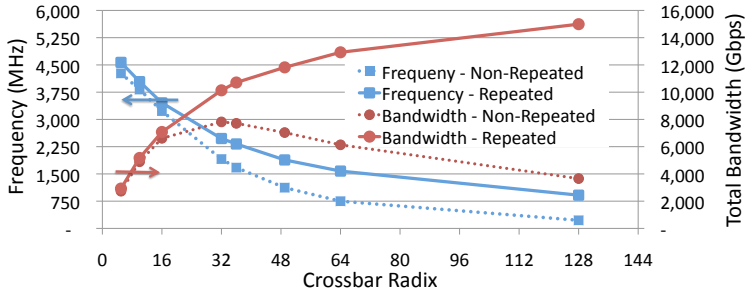


Figure 3.15: Bandwidth and Speed of a *Swizzle-Switch* with 128-bit busses in 32nm. Both repeated and non-repeated versions are presented. When using repeaters the *Swizzle-Switch* scales to designs as large as 128x128x128 resulting in 15 Tbps of total bandwidth.

This study then analyzes two crossbar designs: one with single-segment word-/bit-lines and another with optimally-spaced repeaters. Metal wire delay scales quadratically with distance while repeated wire delay scales linearly. Thus, for high crossbar radices, adding repeaters drastically reduces word- and bit-line delay at the cost of increased power.

Figure 3.15 shows total crossbar bandwidth and maximum frequency as a function of radix, with and without repeaters, for a design with 128-bit busses. A crossbar of radix 64×64×128 supports a bandwidth of 13 Tbps and can operate at greater than 1.5GHz with an area of about $1mm^2$.

3.6 Conclusions

Because of wiring and power challenges in conventional crossbars, high-radix crossbar systems are often seen as infeasible designs. However, the architecture of many-core systems is greatly simplified as the radix of the switches in the interconnect increase.

This chapter details the design of a feasible, high-radix crossbar called the *Swizzle-Switch*. The *Swizzle-Switch* optimizes the arbitration overheads in traditional crossbars by integrating arbitration within switch crosspoints while simultaneously enabling the implementation of fair and prioritized algorithms. It is shown that a *Swizzle-Switch* implementing

LRG arbitration reduces the worst-case request latency by 1.83× and 2.03× on average over Round-Robin and Random arbitration policies, respectively.

The *Swizzle-Switch* design is also scaled to 32nm and shown to enable attractive crossbar design points such as a radix-64 crossbar operating at a frequency of 1.5GHz. The following chapter capitalizes on the benefits of the *Swizzle-Switch* to build a 64-core many-core system called the *Swizzle-Switch Network*.

CHAPTER IV

Swizzle-Switch Networks for Many-Core Systems

Using the optimized crossbar architecture from Chapter III and insights derived from coherence traffic classification, this chapter presents a 64-core crossbar-based, many-core architecture called the *Swizzle-Switch Network* (SSN). The SSN represents a system previously thought to be impractical [66, 96] due to power and wiring constraints.

The SSN is initially compared against a 64-core Mesh topology in order to evaluate the benefits of the SSN against a conventional Network-On-Chip (NoC) architecture. Results show that a many-core SSN can provide significant performance, quality-of-service, and power advantages over a Mesh network of up to 64 cores.

To further motivate the utilization of high-radix switches in many-core systems, this chapter also presents the design of a Flattened Butterfly (FBFly) interconnect optimized by the use of *Swizzle-Switches* inside it's network routers. Flattened Butterfly many-core systems also provide performance and quality-of-services advantages over the Mesh but suffer a power disadvantage relative to the SSN architecture.

The insights gained from this chapter suggest that many-core chip architects should consider flat, crossbar systems as feasible, high-performance designs for systems up to 64 cores and that high-radix topologies such as the Flattened Butterfly should be considered as scaling trends push many-core systems past the 64 core benchmark. This chapter's research is done in collaboration with Ronald Dreslinski, Nathaniel Pinckney and Reetuparna Das.

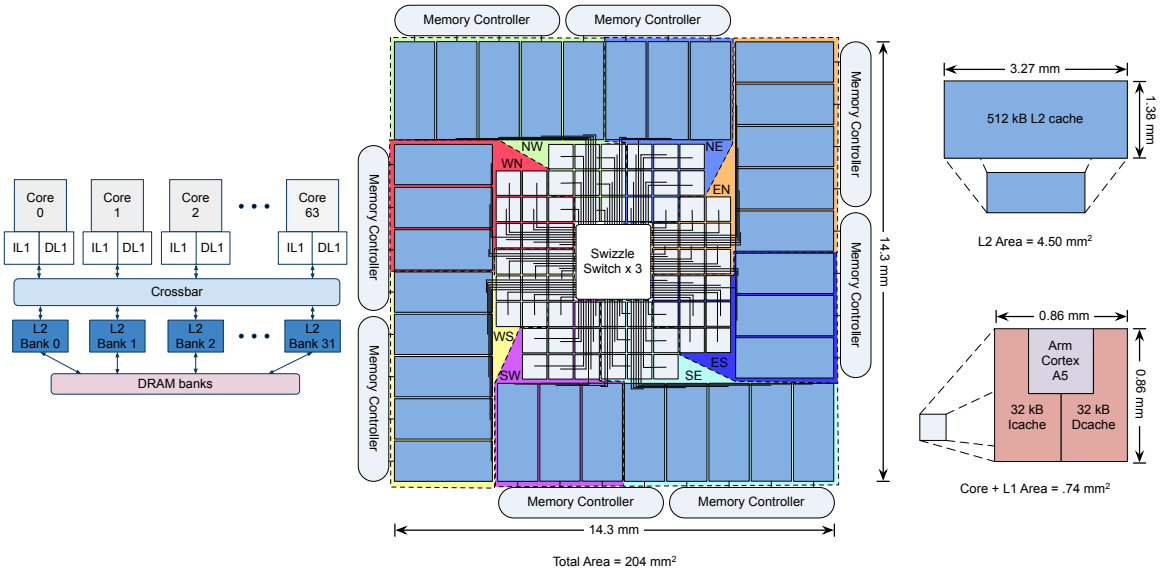


Figure 4.1: (a) High-level architecture diagram (a) of the 64-core *Swizzle-Switch Network* (SSN) built with *Swizzle-Switch* crossbars. (b) The floor-plan of the (SSN) system and estimated dimensions. Octants are colored to aid the reader in seeing how wires leave the crossbar. The total chip area is $204mm^2$, each core/L1 tile consumes $0.74mm^2$, the L2 tiles consume $4.5mm^2$ and the *Swizzle-Switch* consumes $6.65mm^2$.

4.1 Interconnect Design Components

The three interconnect designs in this chapter, the *Swizzle-Switch Network* (SSN), Flattened Butterfly and Mesh, are all targeted in an industrial 32nm process. To provide the most realistic comparison possible, the interconnect topologies are floor planned to target a $200mm^2$ chip. Results from floor planning are then used to drive the simulation models describe in section 4.5.

Cores are modeled after the published characteristics of an ARM Cortex-A5 [4]. Core size, speed, and power are then scaled to 32nm. The 32nm A5 achieves a frequency of 1.5GHz and occupies $0.18mm^2$.

Cache area, latencies, and power are calculated using Artisan SRAM compiler estimates and SPICE simulations. Each 64-core system uses eight interleaved memory con-

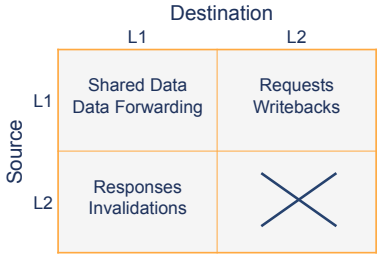
trollers. Additionally, L2 address ranges are assigned to the nearest memory controller in order to minimize interconnect congestion.

The target 32nm process provides a 9-layer metallization stack to utilize in each floor plan. In this metallization stack there are four *1X*, two *2X*, two *4X* and one *8X* metal layers. The *1X* metal layers and one of the *2X* metal layers are reserved for local routing (within the core/cache). The *8X* layer is reserved for power and clock routing. That leaves one *2X* and two *4X* layers for global routing. The interconnect for the NoC and SSN uses only parts of one *2X* and parts of one *4X* layer. Wire delays were determined using wire models from the design kit using SPICE analysis including repeaters, taking into account cross-coupling capacitance of neighboring wires and metal layers. For interconnect wires, four options are considered that trade off area for speed. These options include a *4X* or *2X* metal layer with either single or double spacing. Repeater insertion is adjusted so that repeaters are placed in the gaps between cores. The repeater placement was considered for all topologies to accurately estimate timing. The resulting wire delays ranged from 55-350ps/mm depending on repeater placement, wire spacing, and metal layer.

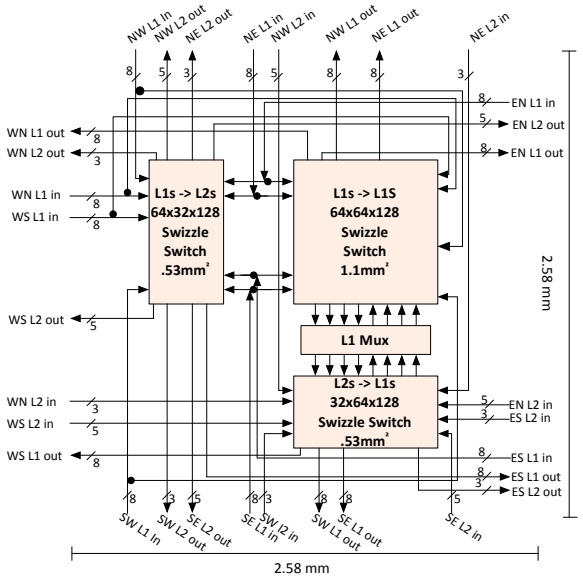
4.2 Swizzle Switch Network

The *Swizzle-Switch Network (SSN)* combines novel circuit and architectural insights to challenge the conventional scaling limitations of crossbars. Crossbar-based systems are desirable in many-core chips because they can ease the burden of managing highly variable memory access latencies. Thus, an SSN-based system can take advantage of new crossbar technology to provide uniform memory access at the many-core level.

Figure 4.1(a) shows one possible configuration of a SSN system. The SSN connects 64 cores, each with their own private L1 instruction and data caches, to 32 banks of L2 cache. The L2 cache banks are then connected to DRAM. Figure 4.1(b) shows one potential layout of such a system. Each tile comprises an ARM Cortex A5 and 32kB L1 I/D caches. The L2 cache is banked in 32 tiles of 512kB each and placed around the perimeter of the cores.



(a)



(b)

Figure 4.2: (a) Classification of communication messages required for coherence (b) Wiring diagram for combining three *Swizzle-Switches* into a 64×64×128bit crossbar. The wires are labeled by the quadrant to which they connect. Each wire in the diagram represents either 3, 5, or 8 busses, where each bus is 128-bits. The overall area of the Crossbar is $6.65mm^2$($\sim$ 4% of the 64 tile system).

Memory controllers are placed in the periphery and are directly connected to the L2s. The colored octants indicate the association of L2 and core tiles to the centralized switch.

4.2.1 Coherence Protocol

A novel design aspect of the SSN is it's use of three *Swizzle-Switches* to enable a directory-based, MOESI coherence protocol. MOESI coherence requires an interconnect fabric to facilitate communication among the private (L1s) and shared (L2s) caches in the system. Figure 4.2 (a) classifies coherence messages into four types: L1→L1, L1→L2, L2→L1 and L2→L2. Note that MOESI protocols require no L2→L2 communication. Consequently, the SSN is optimized to provide only the three required communication paths, each via dedicated *Swizzle-Switches* (omitting a crossbar for L2→L2 communication). This optimization reduces SSN power requirements by 17% relative to a switch with

all four communication paths.

In addition, the multicast ability of the SSN facilitates further traffic and power optimizations for invalidation messages. Within the SSN, the invalidations can be multicast to several L1 caches simultaneously (i.e., driving the SSN input bus only once). In contrast, NoC designs either transmit individual invalidation messages per destination or must employ sophisticated control policies to enable broadcast/multicast [53].

4.2.2 Timing and Layout Evaluation

Figure 4.2 (b) illustrates the layout of the three *Swizzle-Switches* needed to build a complete SSN crossbar: two for the bi-directional interconnect from L1s→L2s and one for communication of shared data from L1s→L1s. Each depicted wire represents several 128-bit buses. Busses are grouped by the chip octant to which they are routed, as identified in Figure 4.1. The diagram reflects the relative locations of input/output busses and corresponds to the floorplan in Figure 4.1.

To calculate the area of the SSN itself, several measurements are needed. The area of each *Swizzle-Switch* is determined from detailed layouts: the two smaller *Swizzle-Switches* occupy $0.53mm^2$ while the larger requires $1.1mm^2$. Routing over the *Swizzle-Switch* is not possible (as it occupies all global-routing metal layers), so all busses that pass around the *Swizzle-Switches* consume area.

Overall, including these route-around overheads, the *Swizzle-Switch* network is $2.58mm$× $2.58mm$, for a total area of $6.65mm^2$ ($\sim$4% of total chip area). The SSN layout also includes some empty space at the periphery due to symmetry/routing constraints, however, this empty space could be used for other circuitry. Considering this additional overhead, the total area of the SSN-based chip is $204mm^2$, a 7% increase in area over the Mesh and Flattened Butterfly topologies.

Interconnect signals take one cycle to reach the crossbar, one cycle to arbitrate, one cycle to pass data through the crossbar, and one cycle to reach the destination. Global

wires are routed in a mixture of *2X* and *4X* metal depending on the distance routed. As a result, the longest wires operate at 1.7GHz. At the most routing dense point in the layout, just outside the Swizzle-Switch, the *4X* metal layer utilization due to SSN routing is 60% of routing tracks and the *2X* metal layer utilization is 40% of routing tracks. In other parts of the chip, routing density due to SSN routing drops substantially and is not a significant factor. From Chapter III Figure 3.15, the 64x64x128 repeated crossbar configuration was selected to maximize bandwidth while achieving a 1.5 GHz clock frequency to match the cores.

4.2.3 Reliability

The SSN's monotholic design can potentially make it susceptible to reliability issues if one or more of the input/output ports experiences a fault. Modifications to enhance the robustness of an SSN are available because of it's SRAM-like layout and it's small total area. Error correction techniques (e.g., ECC) that are used in standard SRAM systems can also be used to enhance SSN reliability. Additionally, redundant output ports could be used to aid fault recovery mechanisms.

4.3 Mesh Topology

The SSN is contrasted against a Mesh topology, which has been used in a number of recent many-core designs [120, 45]. A Mesh is amenable to a tiled design, is easy to lay out, and does not require any long, cross-chip wires. Moreover, by distributing L2 cache slices in each core tile, Meshes can facilitate L2 cache designs that provide low latency from the core to its associated L2 slice. Despite these advantages, Mesh interconnects do not always scale well because they are vulnerable to issues such as router congestion, network power, and non-uniform (and high worst-case) access latencies to remote L2 banks. In addition, bursty and hotspot traffic patterns that often arise in real applications can lead to

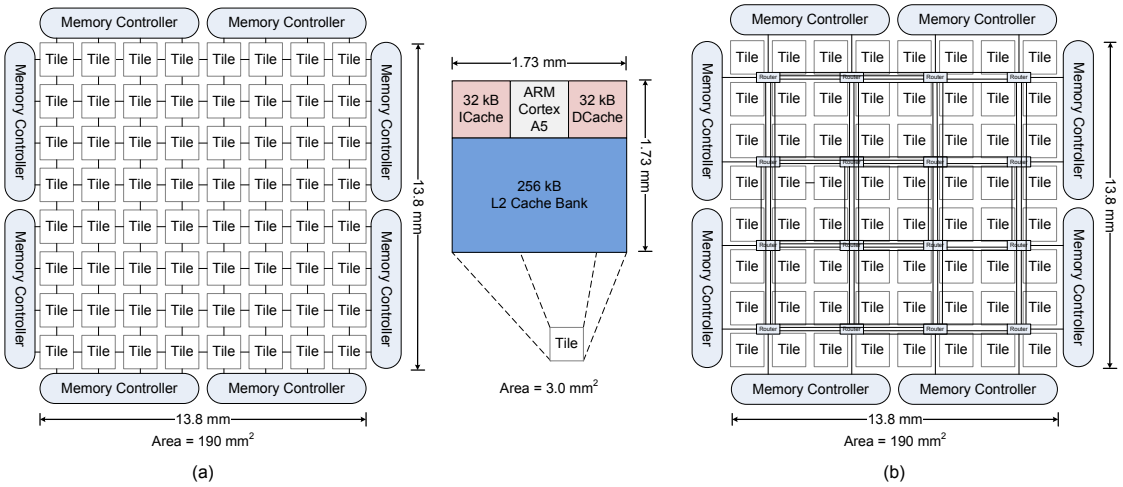


Figure 4.3: Floor-plan of the Mesh and Flattened Butterfly systems with estimated dimensions. The total size of both chips is $190mm^2$.

high queuing delays even if overall interconnect utilization is relatively low.

Figure 4.3 (a) shows the 64-core Mesh layout. Each tile comprises one ARM Cortex-A5, private 32kB instruction and data caches, a 256kB slice of the shared L2 cache, and a router that links the tile to its four nearest neighbors. Tiles at the periphery connect to the memory controllers. The Mesh topology implements 4-stage pipelined routers, using lookahead routing to eliminate the need for a route computation stage [34]. To prevent deadlock, the Mesh utilizes XY-dimension ordered routing and implements 3 virtual networks (request, response, writeback) over 1 physical network. Finally, three virtual channels per virtual network are allocated to reduce network congestion (e.g., due to head-of-line blocking).

The area of each tile is $3mm^2$, resulting in a total chip area of $190mm^2$ (excluding memory controllers). Router latency is dominated by virtual channel allocation and arbitration time, resulting in a peak frequency of $\sim$3.5 GHz. The interconnect links (channels) are 16 bytes wide and $1.73mm$ in length. When routed in *2X* double-space metal they can achieve a speed of 3.1GHz. As such, links are operated in the NoC at 3GHz to match an even multiple of the core frequency.

4.4 Flattened Butterfly

As a second baseline for comparison, this work designs a flattened butterfly network (FBFly) [61]. Recent work has demonstrated that this topology can outperform meshes due to decreased hop count between any tile pair. For example, a 4-ary, 3-flat FBFly can support 64 cores while *bounding* the router hop count to 2 (as opposed to a 64-core Mesh *average* hop count of 8). The FBFly requires considerably higher radix routers than the Mesh. Large high-radix routers are typically slow, however, *by using a* Swizzle-Switch *as the crossbar element within the router*, the design of a FBFly can operate at a 40% higher frequency compared to a conventional crossbar-based router design. Thus, the *Swizzle-Switch* demonstrates its utility as a *building block* for high-radix network design.

Figure 4.3 (b) shows the proposed layout of the Flattened Butterfly topology. Tiles in the FBFly are identical to those in the Mesh. In this layout, each of the 16 routers are connected to 4 tiles creating a 64 node network. The radix for each router in the FBFly can be either 14 or 16 because there are 3 links for routers in the same row, 3 links for routers in the same column, 4 links to the local L1s, 4 links to the local L2s, and up to 2 links to memory controllers.

Although adaptive routing can take advantage of the FBFly's path diversity, implementing such a technique typically includes a signficant amount of router complexity. This work simplifies FBFly routing by always routing to the XY-dimension ordered path that requires 2-hops from source to destination.

The links to nearest-neighbor routers are $3.46mm$ and are routed in the *4X* double spaced metal. These nearest-neighbor links can operate faster than 3GHz. For links to non–nearest-neighbor routers, the wire lengths are either $6.92mm$ or $10.38mm$. These links are both routed in 4x double spaced metal, resulting in up to 620ns of delay. To allow the network to operate at 3GHz, these links are pipelined in 2 stages.

4.5 Simulation Methodology

Table 4.1: *gem5* 64-Core Simulation Parameters

Feature	Mesh & Flattened Butterfly	*Swizzle-Switch Network*
Processors	64 in-order cores, 1 IPC, 1.5 GHz	
L1 Caches	32kB I/D Caches, 4-way associative, 64-byte line size, 1 cycle latency	
L2 Caches	Shared L2, 16 MB, 64-way banked, 8-way associative, 64-byte line size, 10 cycle latency	Shared L2, 16MB, 32-way banked, 16-way associative, 64-byte line size, 11 cycle latency
Interconnect	3.0 GHz, 128-bit, 4-stage Routers	1.5 GHz, 64x32x128bit Swizzle Switch Network
Main Memory	4096MB, 50 cycle latency	

Table 4.2: SPLASH2 Benchmarks tested and input sets

Benchmark	Input Set	Bench.	Input Set
Cholesky	tk15.O	Ocean*	258x258 ocean
FFT	64K points	Radix	1M ints, rad. 1024
FMM	16K particles	Raytrace	teapot
Lu*	512x512 matrix, 16x16 blocks	Barnes	head
		Water*	512 molecules

Each interconnect design in the previous sections are evaluated using detailed timing simulation with the *gem5* full-system simulator [9].To simulate the many-core systems, *gem5* is extended to model the three interconnects and a MOESI directory-based coherence protocol.

Simulation parameters are configured using timing characteristics derived from the SPICE and layout analysis discussed in the previous sections. Table 5.3 details the simulation parameters. To account for non-determinism in threaded workloads, the simulations randomly perturb memory access latencies and are run multiple times to arrive at stable runtimes (as described by Alameldeen *et al.* [2]).

Benchmarks from the SPLASH2 [122] suite are used fore evaluation. The SPLASH2 benchmarks are of particular interest for the study of on-chip interconnects as they have diverse sharing and data migration patterns between cores as shown by Barrow-Williams *et al.* [8].

4.6 Performance Analysis

The *Swizzle-Switch Network* (SSN), Mesh, and Flattened Butterfly (FBFly) systems are evaluated according to four metrics: overall runtime performance, average miss latency,

Table 4.3: Cache Miss Rates and L1 Miss Latency (in CPU cycles)

Benchmark	L1 MPKI	L2 MPKI	L1 Miss Latency to an On-Chip† Location						Speedup†† over Mesh	
			Mesh		FBFly		SSN			
			Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	FBFly	SSN
Barnes	6.2	0.5	53.6	16.1	32.0	5.5	27.1	2.2	1.12×	1.15×
Cholesky	2.4	1.2	57.2	16.7	32.3	5.8	24.7	5.3	1.04×	1.07×
FFT	4.4	1.4	57.5	16.7	33.2	6.7	27.2	7.3	1.11×	1.14×
FMM	2.5	0.7	55.4	16.5	32.1	5.7	25.9	3.9	1.11×	1.15×
LuC.	1.2	0.7	57.1	16.4	32.0	5.6	23.9	5.3	1.02×	1.03×
LuNonC.	1.8	2.0	60.5	16.1	32.3	7.4	22.1	9.6	1.04×	1.05×
OceanC.	17.9	7.1	54.7	16.2	32.5	6.6	26.4	6.0	1.29×	1.43×
OceanNonC.	27.2	8.2	54.2	16.0	32.3	6.0	26.7	5.1	1.31×	1.45×
Radix	22.1	8.6	54.4	16.1	27.2	10.3	26.8	4.3	1.28×	1.37×
Raytrace	7.7	2.0	56.2	16.7	32.6	6.0	25.9	3.8	1.57×	1.82×
WaterNSq.	3.3	0.5	54.4	16.2	32.4	5.9	26.9	4.3	1.05×	1.07×
WaterSp.	0.6	0.2	56.9	16.6	32.5	6.8	25.0	7.8	1.02×	1.02×
Mean			**56.2**	**16.4**	**31.9**	**6.5**	**25.6**	**5.4**	**1.15×**	**1.21×**

†Excludes main memory accesses ††Speedup uses geomean

miss latency variation, and energy/power. The analysis shows that both of the topologies enabled by a high radix crossbar—the FBFly and the SSN—perform noticeably better than the Mesh even though the SSN runs at only half the frequency. The FBFly is 15% faster in overall runtime, has a 1.76× reduction in average L1 on-chip miss latency, and experiences a 2.52× reduction in the standard deviation of L1 on-chip miss latency compared to the Mesh. The SSN has a 33% lower interconnect power, decreases the run time by 25%, reduces the average L1 on-chip miss latency by 2.2×, and provides a 3× reduction in the standard deviation of L1 on-chip miss latency relative to the Mesh.

Table 4.3 shows the speedup for each benchmark and Figure 4.4 shows execution time breakdowns comprising three categories: core active cycles, memory stall cycles, and synchronization stall cycles. From the results, one can observe three different performance-impact scenarios in the results. The first arises for benchmarks with high L1 miss rates and substantial sensitivity to L2 access stalls. *OceanContig*, *OceanNonContig*, and *Radix* all have high L1 Misses Per KiloInstruction (MPKI) as shown in Table 4.3 and also spend a substantial fraction of execution time on memory stalls as shown in Figure 4.4. The *Swizzle-Switch*-based topologies substantially accelerate these workloads, due to the improved average L2 access latency.

The second class of workloads, including *Raytrace* and *FMM*, spend a large fraction of

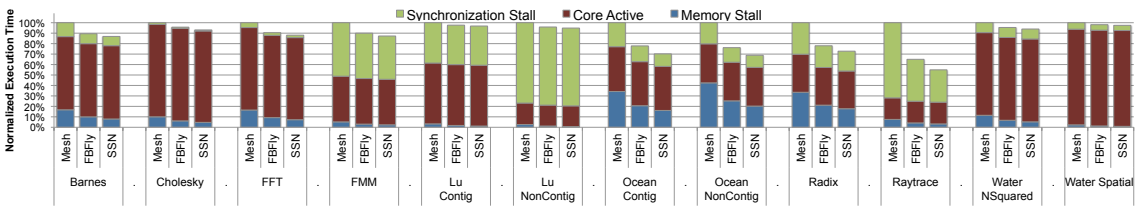


Figure 4.4: Cycle Analysis for 64 core Mesh, FBFly, and SSN topologies during parallel regions of the SPLASH2 benchmarks.

time in synchronization stalls. These particular benchmarks have locks that are sensitive to miss latency. As average miss latency improves, synchronization time is also reduced, yielding significant speedups. When synchronization stalls arise due to load imbalance, as in *LuNonContig*, there is no significant speedup since improving memory latency does not resolve the load imbalance.

The last scenario arises for benchmarks with a low L1 MPKI, for example, *WaterSpatial*. Such benchmarks are insensitive to L2 latency as their working sets fit in L1, and thus, achieve only minimal performance gains (2%) from the faster interconnects.

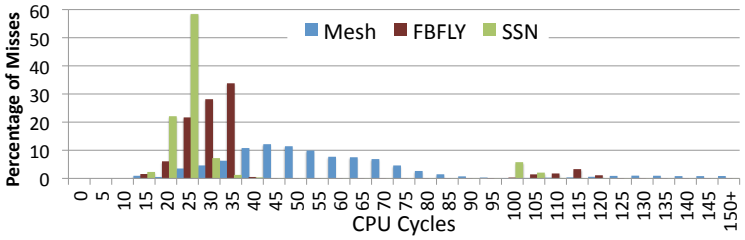


Figure 4.5: Histogram of L1 cache miss latency for the Radix benchmark.

In Figure 4.5, the miss latency distribution is shown for the Radix benchmark (other benchmarks have similar miss latency distributions). Within this distribution, accesses with latencies from 10-80 cycles are serviced on chip, whereas those with latencies above 100 cycles are misses to main memory. The figure illustrates that the high-radix interconnects achieve tighter latency distributions for on-chip accesses. The wide latency variance in the Mesh is due to the highly-variable hop count (as many as eight hops) for some messages. In contrast, messages on the FBFly require at most two hops, while the SSN requires only one

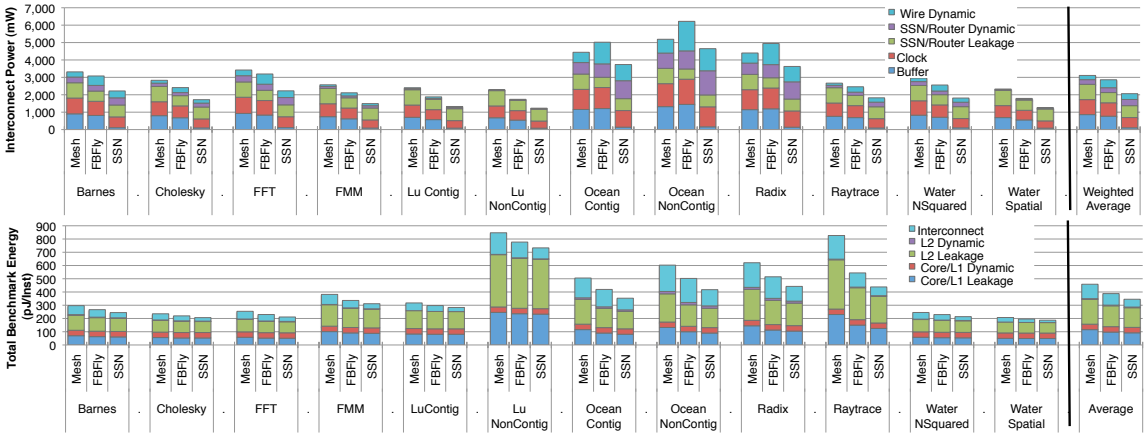


Figure 4.6: Total interconnect power (top) broken down by components within the Mesh, FBFly, and SSN systems for all benchmarks tested. Total system energy (bottom) for each benchmark broken down by component. Overall the SSN reduces interconnect power by 33% over the Mesh and 28% over the FBFly on average. As a result of the lower interconnect power and better performance the total SSN system energy is 25% less than the Mesh and 11% less than the FBFly.

(variance arises only due to endpoint queueing and latency differences between L1-to-L1 and L2-to-L1 transfers). The average and standard deviation of the miss latency is given in Table 4.3. While the Mesh maintains a standard deviation of L1 miss latency of 16.4 cycles, the FBFly is able to achieve a standard deviation of 6.5 cycles and the SSN tightens the standard deviation even further to 5.4 cycles. The more predictable access latency in the FBFly and SSN interconnects makes it easier for programmers to analyze performance, balance work across cores and reduce the need for careful on-chip data placement.

Overall, it has been shown that high-radix topologies enabled by the *Swizzle-Switch* scale well to 64 cores and achieve significant speedups over a Mesh. The FBFly on average sees a 15% speedup over the Mesh, while the SSN further increases the average overall speedup over the Mesh to 21%.

4.7 Energy and Power Analysis

Previously, one of the main criticisms of high radix crossbars was their high power consumption [66]. However, the *Swizzle-Switch Network*'s optimized design demonstrates that high-radix interconnects can be more power efficient than low-radix topologies, which require a larger number of routers and buffers. Figure 4.6 shows the power consumption for the three interconnects broken down into switch, buffers, link, and clocking power sub-components. The Mesh has the highest interconnect power consumption. The FBFly, on the other hand, runs at the same high frequency as the Mesh and has pipeline buffers on its longer wires, therefore requiring higher dynamic wire power than the Mesh but has lower overall router power consumption. The SSN trades off reduction in buffer power with increase in wire dynamic power while maintaining similar switch power. Overall, the SSN reduces interconnect power by 33% over the Mesh and 28% over the FBFly on average.

The total system energy consumption of the SSN is reduced because of the overall runtime improvement. The energy savings arises from conserving leakage energy in the core and L2, which shrink as performance increases and total runtime is reduced. From the plot one can see that the largest improvements in energy consumption correlate to the benchmarks with the largest performance improvement (i.e., *Raytrace*). As a result of the lower interconnect power and reduced runtime, the total SSN system energy is 25% less than the Mesh and 11% less than the FBFly.

4.8 Sensitivity Analysis

To further explore tradeoffs in many-core system design, Sensitivity analysis is performed on 4 key parameters: router pipeline depth, virtual channel sizing, interconnect frequency, and out-of-order core traffic.

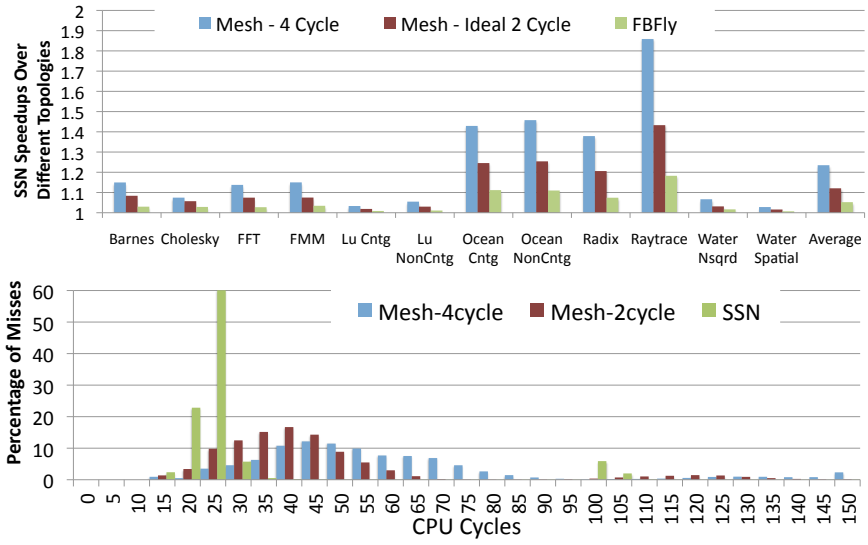


Figure 4.7: Sensitivity analysis using ideal, 2-stage speculative routers. Histogram of the L1 miss latencies for the Radix benchmark.

4.8.1 Router Pipelines

Figure 4.7 contrasts the performance gains of SSN over a Mesh interconnect with 4-stage router pipeline and an idealized 2-stage router pipeline (employing pipeline bypassing and speculative virtual channel allocation) [91]. An idealized design that assumes bypassing is always possible and speculative virtual channel allocation never fails (i.e., by configuring the simulation with a 2-stage pipeline). Even under these idealized assumptions, the 2-stage Mesh still incurs a wide variation in miss latencies and the SSN still outperforms it by an average of 12.4%. Note that this estimate is conservative: gains would be higher when compared to an accurate implementation of a speculative router (due to mispeculation stalls). This section does not consider speculative routers for the FBFly topology because their higher radix make routing implementations considerably more difficult.

4.8.2 Virtual Channels

Figure 4.8 illustrates the impact that the number of virtual channels has on the Mesh interconnect. In this study, three virtual networks (request, response, writeback) are still

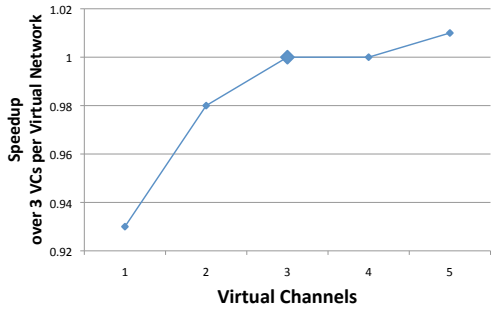


Figure 4.8: Mesh sensitivity to the number of virtual channels (VCs) per virtual network for the Raytrace benchmark. For this example, there is only a 1 % performance improvement using 5 VCs (over 3 VCs) per virtual network. The enlarged data point represents the configuration used in Section 4.6.

needed to prevent deadlock but the number of virtual channels per network is varied. Because the Raytrace benchmark is most sensitive to synchronization, it best shows the significance of virtual channel allocations for the Mesh. It is observed that simply using 3 virtual networks with 1 virtual channels can cause up to a 7% degradation in performance, while the benefit of additional virtual channels stagnates at around 3 virtual channels. Consequently, the 3 virtual channels per virtual network design represents a good design choice.

4.8.3 Interconnect Frequency

Figure 4.9 evaluates the performance of the SSN, the 4-cycle Mesh and the ideal 2-cycle Mesh when the interconnect frequency is varied from 1.5GHz to 6.0GHz. To isolate the impact of the interconnect, the cores held at a constant 1.5GHz for all design points. As expected, a flat crossbar system is always advantageous to a Mesh when the interconnect frequencies are equal. However, the study shows that the average number of router hops in the network-on-chip system is such a prohibitive factor that it would take a Mesh running at 6.0GHz to match the performance of a SSN running at 1.5GHz. Consequently, one can see that the benefit of building a flat system can be significant given the feasiblity constraints of building a network-on-chip with an aggressive clock as well as small number of pipeline stages.

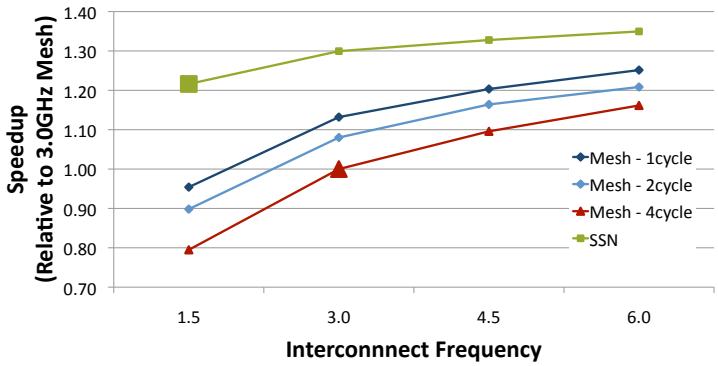


Figure 4.9: Sensitivity to the interconnect frequency for the Mesh and SSN (Cores remain at 1.5GHz). Results show that a Mesh w/4-cycle routers needs to be run at 4x the frequency of a SSN to achieve the same performance. The enlarged data points for the SSN and Mesh represent the configurations used in Section 4.6.

4.8.4 Out-of-Order Cores

The performance impact of using out-of-order cores on a 64-core system is demonstrated in Figure 4.10. Replacing the in-order cores with out-of-order cores would present significant area and power overheads to both the SSN and NoC systems. The increased wire length between the SSN and the L2 caches would cause the SSN to be clocked at a lower frequency. Similarly, the increased distance between connected routers in the NoC topologies would also present frequency degradation as well as increased buffer sizing demands. For the purposes of this example, those limitations are ignored and instead this sensitivity study focuses on generating as much core-stimulated traffic as possible in the interconnection networks. This is done by placing 1.5 GHz, 8-wide out-of-order cores (64 inst. window, 64 load/store queue, and 256 physical regs.) in each network and assuming the frequency of the SSN and NoC topologies are equivalent to their achieved speeds when using inorder cores (1.5GHz and 3.0GHz respectively).

The increased traffic from the out-of-order cores magnifies the workload trends that were seen in Section 4.6. For example, the SSN achieves a 7% speedup over the Mesh on the compute-intensive WaterNSquared when in-order cores are used on both systems,

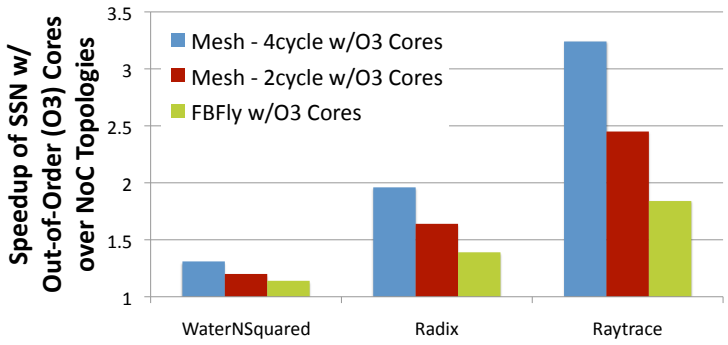


Figure 4.10: Speedups of a 64-core SSN using out-of-order cores over 64-core NoCs also using out-of-order cores. Benchmarks shown represent the 3 traffic classes referenced in Section 4.6. The compute intensive benchmark (WaterNSquared) sees a 1.31x improvement while the memory-intensive (Radix) and synchronization-sensitive workloads see ∼2x and ∼3x improvements respectively when using out-of-order cores.

but sees a 31% improvement when out-of-order cores are utilized. Similarly, the SSN's in-order core speedup of Radix and Raytrace are 37% and 82% respectively over the Mesh. However, those gains become 1.96x and 3.24x speedups when the evaluation is performed only using out-of-order cores. The increased amount of memory traffic from out-of-order cores on gives the SSN more opportunity for optimization. Consequently, it is shown that a flat crossbar system can potentially be of even greater benefit if constructed for systems with high-performance, out-of-order cores.

4.9 Conclusions

The results from this chapter demonstrate that a flat, crossbar-based many-core system can outperform a conventional Network-on-Chip (NoC) many-core when evaluated using performance, quality-of-service, and power metrics. A 64 core *Swizzle-Switch Network*, the many-core system presented in this chapter, has a 33% lower interconnect power, decreases the run time by 25%, reduces the average L1 on-chip miss latency by 2.2×, and provides a 3× reduction in the standard deviation of L1 on-chip miss latency relative to a 64 core

Mesh.

Additionally, a number of sensitivity studies are performed in order to fully understand the tradeoffs of a *Swizzle-Switch Network* against the other NoC topologies. From this analysis, it is shown that the traffic from out-of-order cores can increases the performance benefits of a *Swizzle-Switch Network* to 3× over a 64 core Mesh. These studies also observed that a 64 core Mesh would need to be run at 4× the frequency of the *Swizzle-Switch Network* in order to match performance.

Lastly, this chapter demonstrates that high-radix topologies enabled by *Swizzle-Switch* technology are feasible alternatives to conventional, low-radix NoCs. A 64 core, *Swizzle-Switch*-optimized Flattened Butterfly is 15% faster in overall runtime, has a 1.76× reduction in average L1 on-chip miss latency, and experiences a 2.52× reduction in the standard deviation of L1 on-chip miss latency compared to a 64 core Mesh. These results also suggest that as systems continued to move past 64 cores, high-radix networks can be the building block needed for future scalability.

CHAPTER V

Scalable 3D Interconnects

The previous chapters demonstrated various challenges and solutions for building interconnect architectures to support many-core systems. As the number of cores rises in a system, interconnect scalability is limited by issues such as wiring complexity, total area, and power.

Three-dimensional (3D) integrated circuits are attractive options for overcoming the barriers in interconnect scaling. In a 3D chip, multiple circuit layers are stacked together with direct vertical interconnects tunneling through them. By utilizing the 3rd dimension for core and memory components, 3D architectures present an opportunity for interconnect architectures to efficiently support many-core systems.

This chapter looks at the impact of 3D interconnects on many-core architectures. First, a brief overview of 3D integration is presented. Next, a 3D-*Swizzle-Switch Network* (SSN) is shown to provide further optimizations over the original SSN architecture presented in Chapter IV. Lastly, this chapter shows how 3D-technologies can extend bus architectures to support many-core systems. The work in this chapter was done in collaboration with Ronald Dreslinski and Tom Manville.

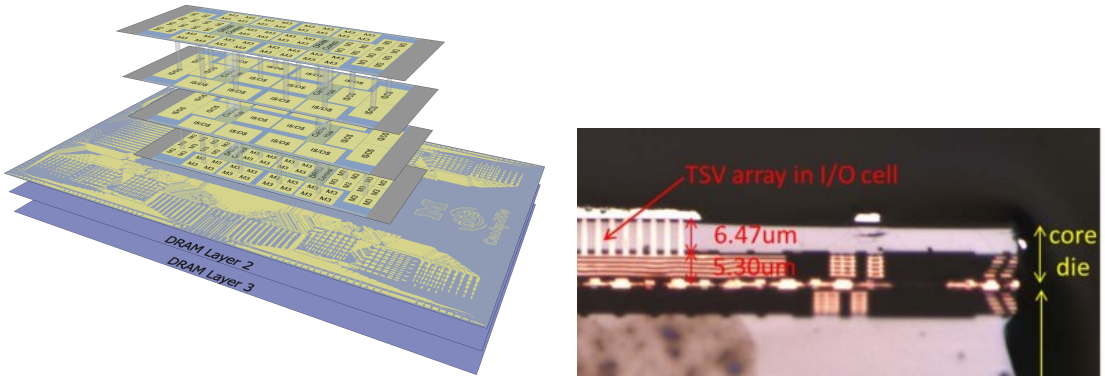


Figure 5.1: Top level view of the Centip3De 7-layer 3D system [33] built on Tezzaron 3D stacking technology and a cross section of the same process on the 3D-MAPS system [59]. Note the TSV's are only 6.47 microns deep and the wafer is thinned to less than 12 microns which is important for reducing thermal resistance and RC delays.

5.1 3D Integration Technology

There have been many works that research 3D technology for use in logic circuits [92], memory optimizations [113, 75] and full-system architectures [57]. 3D integration techniques include die-to-die bonding [12], wafer-to-wafer bonding [74], and various technologies that mix die and wafer integration.

This work focuses on the use of 3D for scalable, high-radix interconnects and assumes the Tezzaron [41] 3D integration which uses a via-first, back-end-of-line integration technology. The technique has been demonstrated in test chips such as centip3De [33] and the 3D-MAPS processor [59].

In Fick *et al*.'s centip3De, the Tezzaron technology is used to stack four layers of CMOS logic on top of three layers of DRAM. Figure 5.1 (left) shows a diagram of the system implemented by Fick *et al*. Figure 5.1 (right) shows a cross section [59] of the Through-Silicon Via (TSV) technology from Tezzaron. The layers are thinned to less than 12 microns, and the TSV's themselves are less than 6 microns thick. The size of the TSVs are 1.4 square microns, and can be placed with a density of 62,000 TSVs per square mm. This technique's high density of TSVs allows architectures to further take advantage of vertical integration

and alleviates some of the concerns of TSV-limited systems detailed in prior works [60, 47].

Additionally, the resistance ($<.35\Omega$) and capacitance ($2fF$) of these TSV's are extremely small compared to other 3D technologies. This allows for fast and short connections between layers that. In fact, in a 4 layer stack the length of a TSV running the whole height of the stack is <50 microns. Along with the reduction of total wiring area, the potential of TSVs to operate within one clock cycle of traditional cores frequencies (1-2GHz) is another added benefit that 3D architectures.

5.2 3D-Swizzle-Switch Networks

The *Swizzle-Switch Network* (SSN) designed in Chapter IV is limited by the overheads of global interconnecting cores and memory in a single, high-radix router. One of the most important benefits of a 3D chip over a traditional two-dimensional (2D) design is the reduction of these global interconnects. This section proposes a combination of 3D integration and the SSN to further improve performance over the original SSN design.

5.2.1 Architecture

In the 3D-SSN design, the baseline 2D-SSN is folded over multiple layers while holding the overall design constant at 64 cores. By folding the *Swizzle-Switch* over multiple layers the total size of the *Swizzle-Switch* is reduced and the shorter wires result in lower capacitance. This reduction in capacitance is translated into a speedup of the *Swizzle-Switch* itself. In addition the number of inputs and outputs per layer is reduced, leading to a more compact design where the links to and from the *Swizzle-Switch* are shorter and faster. The basic 3D-SSN design is explored presented in Figure 5.2(a). In the 4-layer system each layer contains 16 cores and 8 L2 banks. The central *Swizzle-Switch* communicates through TSV's to the other layers in the system. The modified circuit diagram is presented in Figure 5.2(b), for illustrative purposes only two inputs, two outputs, and two layers are presented. The system requires that inputs on one layer forward requests to the proper out-

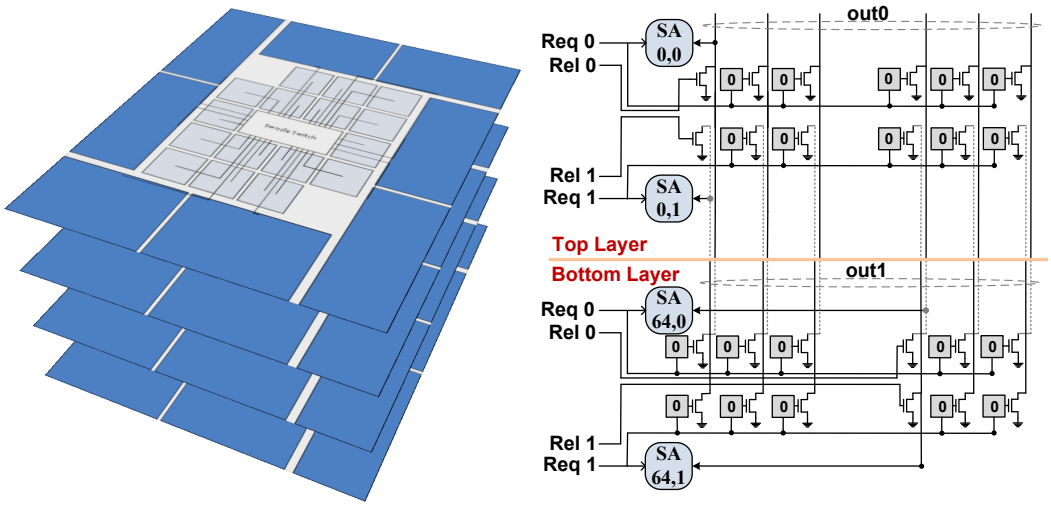


Figure 5.2: (a) A 3D *Swizzle-Switch Network* achieved by stacking four 2D *Swizzle-Switch Network* layers and using TSV's to interconnect the layers. (b) The modified circuits for the 3D *Swizzle-Switch Network*. Even numbered outputs are arbitrated on the top layer, odd outputs are arbitrated on the bottom layer. Input request lines must be forwarded from the top→bottom or bottom→top through TSV connections.

put layer via TSV's. The total number of TSV's at each layer is equal to the total number of arbitrating crosspoints times the number of layers. The exploration of more sophisticated 3D designs is left as future work.

If the total area of the 2D-SSN is split evenly across the 3D layers it will result in a 50% and 75% reduction in area for a 2 and 4 layer system respectively. However, the 2D-SSN is already very dense in both wiring and logic. Currently the 2D-SSN system requires one additional unused routing lane per four lanes of dedicated routing to fit the required logic. The addition of TSV's in the system will dilate the size further. Given the minimum spacing in the Tezzaron process, this corresponds to an additional routing track every 8 tracks in the 2-layer system and every 4 tracks in the 4-layer system. This means the *Swizzle-Switch* in the 2-layer 3D-SSN is 57% of the size of a *Swizzle-Switch* in 2D and the *Swizzle-Switch* in the 4-layer 3D-SSN is 32% of the size of a *Swizzle-Switch* in 2D.

This reduced area of the SSN yields a faster design, and the smaller number of devices

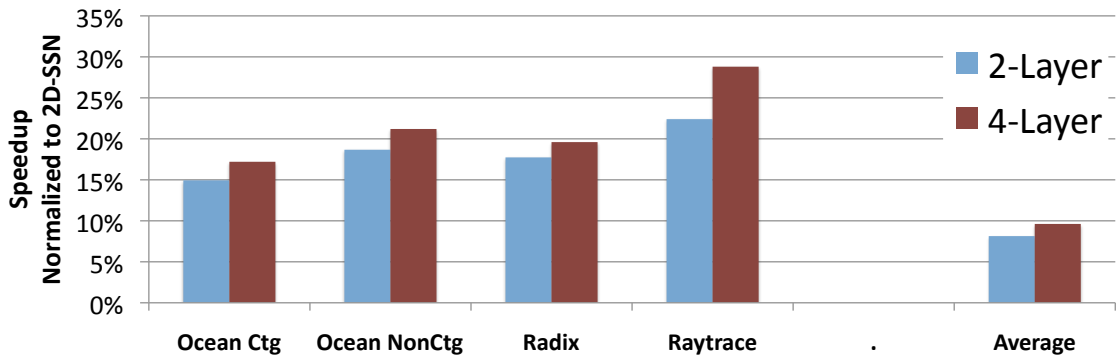


Figure 5.3: Speedup of the 3D-SSN on 2-layer and 4-layer systems compared to a 2D-SSN. The benchmarks most sensitive to interconnect delay are plotted as well as the average across all benchmarks.

per layer shortens the links to the SSN. After careful floorplanning of the system, the 2-layer version achieves an interconnect speed of 2.2 GHz, and the 4-layer version achieves an interconnect speed of 2.7 GHz (core speeds remain at 1.5GHz).

5.2.2 Performance Results

Figure 5.3 presents the overall speedup of the 3D-SSN normalized to the 2D version. On average the 3D system sees a 8% and 10% speedup for a 2-layer and 4-layer system respectively. For benchmarks where the interconnect latency is a significant fraction of the runtime—*Ocean*, *Radix*, and *Raytrace*—the improvements are more significant showing a 15-28% speedup. Recall from Chapter IV that these benchmarks improved significantly due to either faster L2 accesses, or faster synchronization and as such reflect further gains in a 3D design. The other benchmarks only showed marginal sensitivity to interconnect latency and result in only modest gains 1-6%, lowering the average across all benchmarks.

5.2.3 Thermal Analysis

As with any 3D chip design, thermal constraints can be a matter of concern. To verify the system operates in a thermal region that can be cooled by conventional solutions, the

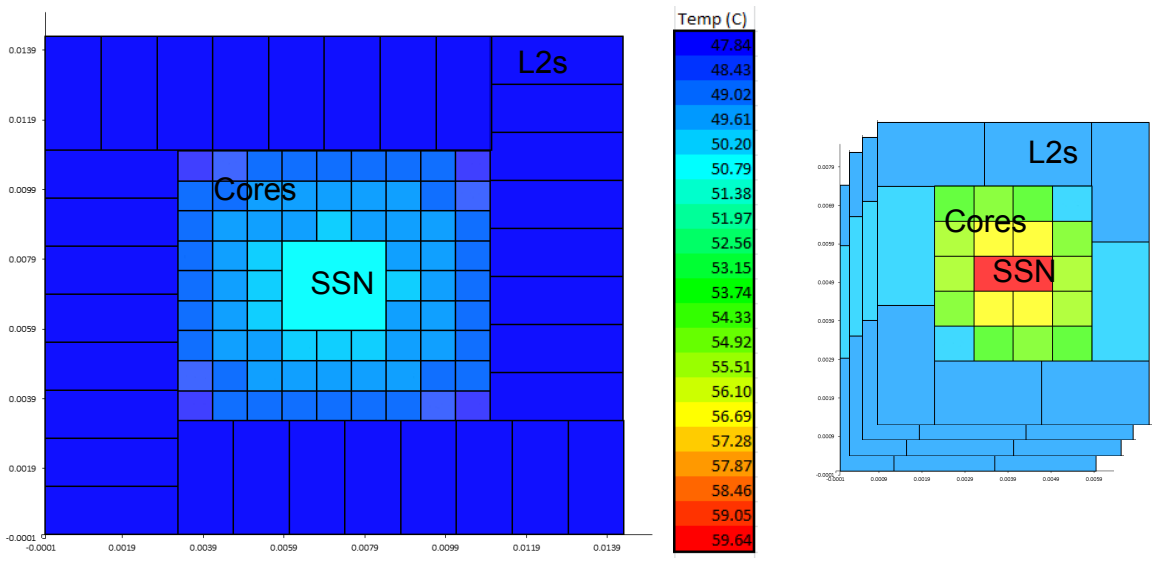
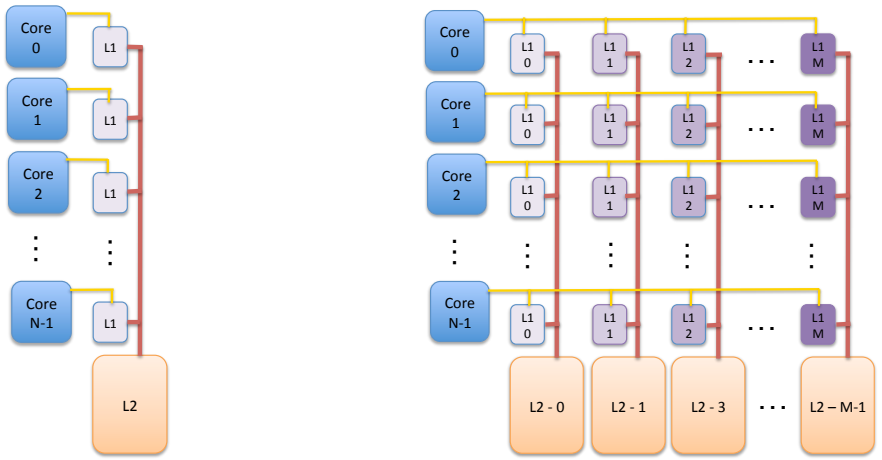


Figure 5.4: HotSpot simulation of 64 Core SSN system on 1 and 4 layers for the worst case benchmark. The peak temperature of the 3D chip is 60^o Celsius.

system is analyzed with the HotSpot 5.1 [107] simulator. The thermal characteristics of the Tezzaron process were modeled in HotSpot and power draw numbers from the hottest benchmark (*Ocean*) are plotted. Power numbers for the Cortex-A5 were based on published data and scaled to 32nm. Figure 5.4 shows the simulated system for a single layer and a 4-layer stack. The low power design of the Cortex-A5 processor helps to make stacking feasible. The peak temperature of the 4-layer system peaks around 60 degrees centigrade, within the capability of passive cooling solutions. The HotSpot analysis did not consider the thermal dissipating characteristics of the TSV's, which would have further reduced the peak temperature.

5.3 XPoint: Scaling Many-Core Busses to 3D

As detailed in Chapter II, bus-based systems traditionally do not scale to many-core systems because of the long wires that prohibit high-frequency designs and contention that limits the parallelism of it's interconnected components. This section proposes XPoint—an bus-based design that minimizes the wire length and contention constraints seen in conven-



(a) (b)

Figure 5.5: High level view of (a) a conventional bus based architecture, and (b) The *XPoint-2D* architecture. Caches in a vertical column are all assigned to the same address range. No snooping is required between vertical columns. The horizontal connections use point-to-point links.

tional designs.

5.3.1 Architecture

As the number of cores grows, the contention on the shared bus becomes a bottleneck in the system. The *XPoint* architectures addresses this bottleneck by banking and interleaving caches in a shared memory many-core system. A significant advantage of this memory interleaving scheme is that the snoopy coherence protocol found in typical bus-based systems does not need to be modified.

This baseline system builds on memory interleaving techniques developed in the 1980's to address congestion in board level multi-processor interconnects [121]. However, integration levels and pin constraints limited their practicality. The proposed *XPoint* systems show that with new architectural techniques these designs are now practical for single chip implementations, where point-to-point connections, higher cache associativites, and a greater number of wires are available.

Figure 5.5(a) shows a high level view of a traditional bus based system. Each core is

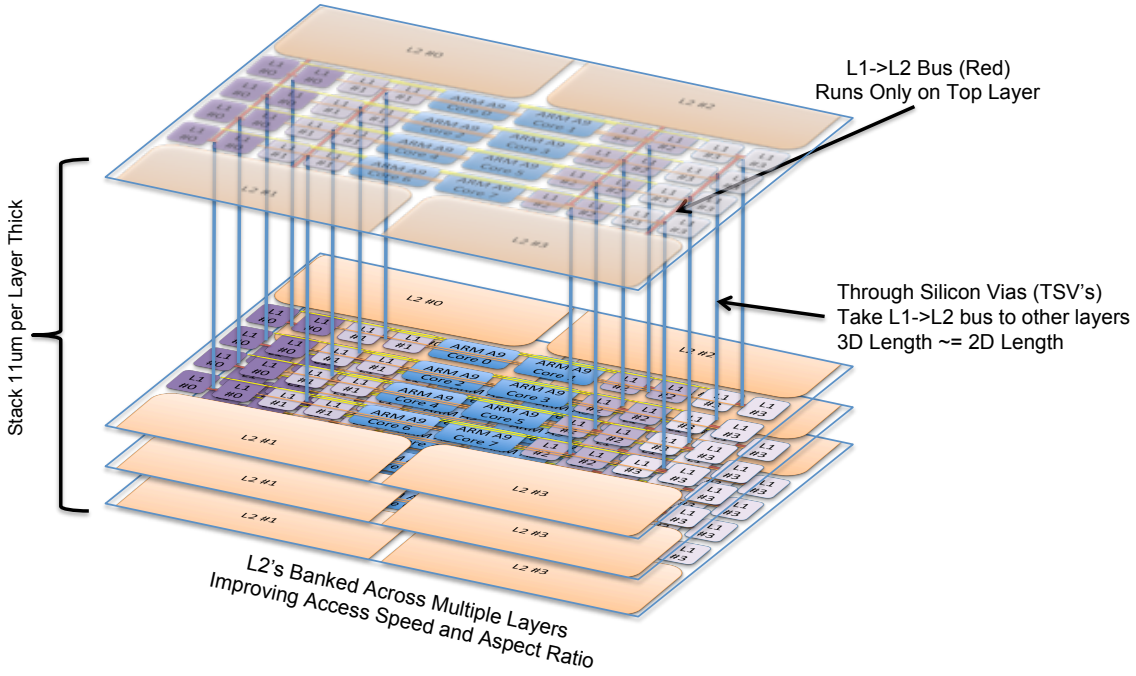


Figure 5.6: Diagram of the *XPoint 3D* design.

connected to a private L1, each of the L1's are connected to a shared snooping bus, which in turn is connected to the L2. The *XPoint 2D* system exploits address-interleaving to reduce bandwidth pressure on the shared bus fabric. Figure 5.5(b) shows the logical layout of the *XPoint 2D* architecture. To scale the coherence protocol to n cores, each core's last level of private cache (L1 in this diagram) is split into m equal slices. The core can access all the m slices via direct point-to-point channels. By using point-to-point connections, the speed of the interconnect can be much faster than a traditional bus which requires bi-directional repeaters and arbitration units. The shared L2 cache is also split into m equal slices. A bus connects a vertical column of private cache slices (n L1 slices) to a shared L2 cache slice, all of which map to the same addresses. This isolates the coherence traffic separately on each vertical bus from the other vertical buses in the system. The result is a multi-bus system with simple, unmodified coherence that reduces contention and increases bandwidth.

Figure 5.6 shows the proposed *XPoint 3D* system. The coherent buses are run only

Table 5.1: Component Areas and Speeds in 32nm

	Area/Length	**Speed/Latency**
ARM Cortex-A9	$0.38mm^2$	$1.25GHz$
L1 (64kB Split I&D) per core	$0.61mm^2$	1 cycle
L2 (256 kB Unified) per core	$1.33mm^2$	8 cycles
Coherent Bus	$0.31mm/core$	160 ps/mm

along the top layer, and connections to L2's at lower levels are made with TSV's which are 11 microns long per layer. This pitchfork layout creates a 3D bus that is approximately the same length as the bus on the top layer. This means that the effective length of the bus is halved when the system contains 2-layers, and quartered when spread across 4-layers. To a first order the latency of the bus grows linearly with $cores/layers$ instead of linearly with $cores$ as it does in a 2D system. In addition the L2 cache is banked across multiple layers to improve the aspect ratio of the cache itself, thereby improving access times and power consumption of the L2.

5.3.2 Methodology

The following text details the design and simulation methodology used to evaluate the baseline bus and *XPoint* systems. This includes methods to calculate area and speed for the core, cache, and bus components. Additionally, parameters and framework for the many-core simulation are discussed.

5.3.2.1 Components

Cache sizes and speeds are obtained from a commercial memory compiler and are listed in Table 5.1. The L1 access time is 1 cycle for the bus based system and 2 cycles in the *XPoint* cache system due to the point-to-point link latency. To hide this latency in the *XPoint* cache system a 4 cache line, fully-associative buffer is added as a L0 next to each core. On a cache lookup, the L0 and L1 are accessed in parallel.

Bus delays and power estimations were obtained using a 13-layer metallization stack

Table 5.2: 2D Floorplan Sizes and Bus Speeds

Number of Cores	Die Size	Bus Length	Bus Speed	Total L2 Size
8	$19mm^2$	$2.48mm$	$>1.25\ GHz$	$2MB$
16	$38mm^2$	$4.96mm$	$1.25\ GHz$	$4MB$
32	$78mm^2$	$10.1mm$	$\sim 630\ MHz$	$8MB$
64	$156mm^2$	$20.1mm$	$\sim 315\ MHz$	$16MB$

Table 5.3: *gem5* Simulation Parameters

Component	**Bus Based System**	***XPoint System***
Processor	ARM Cortex-A9, 1.25 GHz, 2-Wide, 56 Physical Registers	
Cache Block Size	64 Bytes	
L0 Cache	None	4-entry, fully associative, 1-cycle
L1 Cache	64kB Split I and D Caches	
	4-way Associative, 1-Cycle	Divided by number of Slices Associativity 4-way/#Slices, 2 -cycle
L2 Cache	256 kB per Core, 16-way, 8-cycle	
Coherent Bus	312-1250 MHz, 64Bytes	
Main Memory	2GB, 50 Cycle Latency	

from an industrial 32nm process. Once layers were reserved for local and global routing, bus delays were calculated using wire models from the design kit and SPICE analysis. Repeaters are used to achieve optimized frequency on the bus designs and are placed using the optimally interleaved spacing process described by Ghoneima and Ismail [36]. Table 5.2 shows the achieved speeds of the bus for each core count.

5.3.2.2 Simulation

The *Bus 2D* (conventional bus), *XPoint 2D*, and *XPoint 3D* architectures are evaluated using the *m5* full-system simulator [10]. The *m5* simulator was extended to accommodate the busses and point-to-point links of the *XPoint* architecture. Table 5.3 details the simulation parameters for the studies. To account for non-determinism in threaded workloads, randomly perturb memory access latencies are used to run multiple simulations and arrive at stable runtimes (following the approach described by Alameldeen *et al*. [2]). Benchmarks from the SPLASH2 [122] suite to test the systems. The SPLASH2 benchmarks are

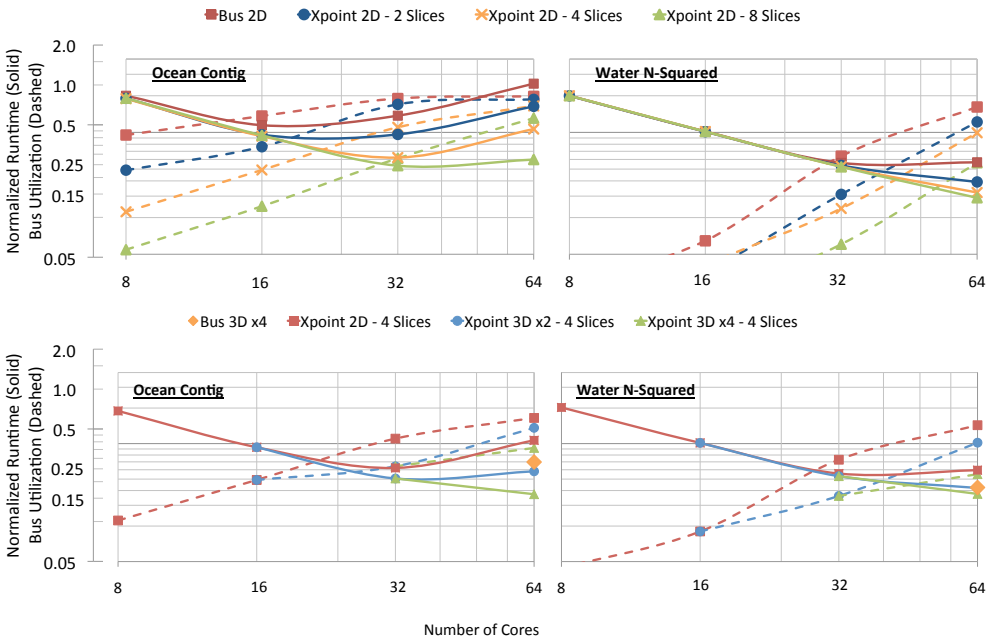


Figure 5.7: Runtime (solid lines) Bus Utilization (dotted lines) *vs*. core counts for Conventional Bus and XPoint Systems. A straight line for runtime represents ideal scaling of the benchmark.

of particular interest for the study of on-chip interconnects as they have many sharing and data migration patterns. This is illustrated in the work of Barrow-Williams *et al*. [8].

5.3.2.3 Thermal Analysis

As in Section 5.2.3, thermal analysis of the system is performed using the HotSpot 5.1 [107] simulator. The thermal characteristics of the Tezzaron process were modeled in HotSpot and peak power draw numbers were used for the core. Power numbers for the Cortex-A9 were based on published data [76] and scaled to 32nm. HotSpot results show that the peak temperature of the 4-layer system reaches 92 degrees centigrade. Thus, even without considering the thermal dissipating properties of the TSVS, the design is still within conventional cooling solutions.

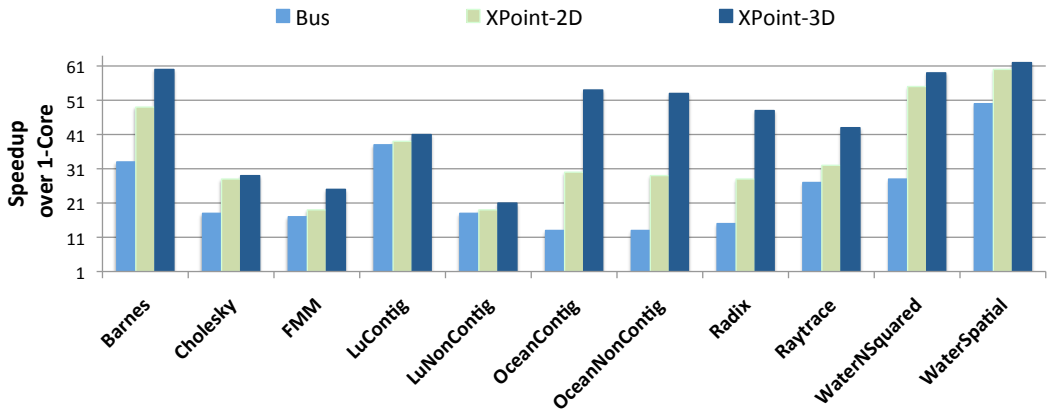


Figure 5.8: Speedup comparison for Bus, *XPoint 2D*, and *XPoint 3D* interconnects. The best performing parameters for each benchmark and configuration are used. Details of number of cores, slices, and layers are found in Table 5.4

5.3.3 Performance Analysis

Analysis of the *XPoint* systems are made through the evaluation of speedup and bus utilization. The *XPoint-2D* system is first evaluated against a conventional bus architecture. Then, it is shown how *XPoint-3D* can push scalability even further past the *XPoint-2D* gains. The *XPoint* systems are also compared against a 3D-conventional bus architecture to further demonstrate the advantages of alleviating contention and latency in many-core bus systems.

The plots in Figure 5.7 show, on a log-log plot, the normalized runtime (solid lines) and bus utilization (dotted lines). The ideal scaling of these benchmarks would be a straight line (for runtime) where the slope is determined by the efficiency of the parallel scaling in the application itself (workload imbalance and/or ratios of serial to parallel code). Bus utilization is a metric from 0 to 1, describing how often the bus is utilized. The bus utilization numbers saturate for the *Bus 2D* (conventional bus) systems for most benchmarks around 32 cores. This saturation correlates with the dramatic increase in bus contention plotted in Figure 2.4. As the bus utilization and contention increase, the overall runtimes of

the benchmarks increase. Eventually the increase in contention outweighs the performance gains of more cores, yielding an optimal number of cores for the system (*i.e.* highest performing system).

The *XPoint 2D* system is most effective for benchmarks with scalability limitations due to contention. As the number of slices (cache banks) in the system grows, the probability of contention decreases (as well as associated snoop overheads). In benchmarks whose *Bus 2D* line is not straight, the *XPoint* cache reduces bus contention and lowers the runtime at high core counts (making the line straighter). For example, when *Water N-Squared* is run on the *XPoint 2D*, 8 slice system the runtime speedup is brought back to linear.

However, *XPoint 2D* is not effective for every contention-limited benchmark. For benchmarks like *Ocean Non-Contig*, the *XPoint 2D* system is only effective up to 32 cores. This limit occurs because the bus latency is increasing as the core count is increased, reducing the total bandwidth of the system and effectively creating more contention.

By extending the system in 3D, the bus latencies are decreased and further scaling is possible for these benchmarks. Figure 5.7 shows runtime and bus utilization on the same axes as before, but for *XPoint 2D* and *XPoint 3D* designs with 4-slices. For the *XPoint 2D* benchmarks that did not scale well—*FMM*, *Ocean Contig*, *Ocean Non-Contig*, *Radix*, and *Water Spatial*—the use of *XPoint 3D* helps reduce the runtime at 64 cores, making the trend closer to linear. On average, the *XPoint 3D* improves performance by 28% over the *XPoint 2D* system at 64 cores.

The best performing configurations for each benchmark on each type of system are shown in Figure 5.8 and listed in Table 5.4. The average speedup achieved by the *XPoint 2D* system is 35× , the *XPoint 3D* system further improves that number to 45×. On average the *XPoint 2D* system performs 1.6× better than the bus, it also outperforms the conventional bus in 3D for all but 3 benchmarks. Overall the *XPoint 3D* system improves performance by 2.1× compared to the 2D conventional bus.

Overall, the results have shown that the *XPoint* architecture scales bus-based systems

Table 5.4: A Breakdown of the best performing parameters on each system for each benchmark. Speedup is presented over a uniprocessor system with 2MB L2. Super linear speedups occur for FFT due to the increased L2 size available in the 64 core system.

	Bus 2D		*XPoint 2D*			Bus 3D			*XPoint 3D*			
	Cores	Sp.†	Cores	Slices	Sp.†	Cores	Layers	Sp.†	Cores	Slices	Layers	Sp.†
Barnes	64	33×	64	8	49×	64	4	58×	64	2	4	**60×**
Cholesky	64	18×	64	8	28×	64	4	27×	64	2	4	**29×**
FFT	32	22×	64	8	59×	64	4	41×	64	8	4	**83×**
FMM	32	17×	64	8	19×	64	4	21×	64	8	4	**25×**
LuC.	64	38×	64	2	39×	64	4	41×	64	2	4	**41×**
LuNonC.	64	18×	64	8	19×	64	4	20×	64	4	4	**21×**
OceanC.	16	13×	32	8	30×	64	4	23×	64	8	4	**54×**
OceanNonC.	16	13×	32	8	29×	64	4	18×	64	8	4	**53×**
Radix	16	15×	32	8	28×	64	4	17×	64	8	4	**48×**
Raytrace	32	27×	64	8	32×	64	4	38×	64	8	4	**43×**
WaterNSq.	32	28×	64	8	55×	64	4	51×	64	8	4	**59×**
WaterSp.	64	50×	64	4	60×	64	4	61×	64	8	4	**62×**
Geomean		22×			35×			31×				**49×**

†Speedup

to 64 core systems. Given these results, *XPoint 2D* presents a desirable approach to interconnect at 32-64 cores in 32nm. As 3D technology becomes part of the commercial mainstream the *XPoint 3D* systems will further scale bus based designs to higher core counts.

5.4 Conclusions

The work in this chapter demonstrated the benefits of using 3D integration technology in many-core interconnects. 3D-integration is demonstrated to allow interconnects to run at higher frequencies as well as alleviate contention across shared interconnect resources (*i.e.*, channels).

First, a 3D-*Swizzle-Switch Network* architecture is presented and shown to operate at 50% and 80% higher frequencies when using 2-layer and 4-layer stacks respectively. The 3D-optimized SSN is able to leverage these frequency gains to achieve 15-28% speedup on memory-intensive benchmarks and 10% average speedup across all benchmarks.

Next, the bus-based *XPoint* architecture is presented as a system that can use 3D technology to effectively scale multithreaded benchmarks. *XPoint* uses 3D to optimize both

contention and latency over conventional bus interconnects. As such, a 64-core *XPoint-3D* system obtains a 49× speedup over the baseline uniprocessor system.

The results shown in this chapter demonstrate that 3D bus and crossbar interconnects are attractive design points for many-core systems. The decision between when to use a 3D bus, crossbar, or NoC interconnect continues to be a challenging one dependent on application domain, topology parameters and circuit limitations. Additionally, simulation frameworks must be developed in order to make a fair comparison between interconnects. As such, the further evaluation of 3D technology past 64 cores is a study left for future work.

CHAPTER VI

Conclusions

As the number of cores on a chip increases, the demand for scalable interconnection networks increases as well. Ideally, scalable interconnects would simultaneously support high bisection bandwidth, predictable memory access latencies, and quality of service guarantees as core count grows. Network-on-Chip (NoC) systems can provide high bandwidth but typically are unable to support applications requiring uniform memory access or quality of service due to high hop counts associated with these networks. Bus- and crossbar-based systems can enable more consistent memory latencies but have traditionally been considered infeasible due to wiring, power and area challenges.

This thesis analyzes the scalability of many-core systems and proposes interconnect architectures that can assist in solving these scalability challenges. Chapters I and II introduce and motivate the need for high-radix interconnect architectures to support many-core systems.

Chapter III continues this thesis' many-core scalability efforts by revisiting the design of crossbar and high-radix interconnects in light of advances in circuit techniques that significantly improve crossbar scalability. A new circuit-level building block, the *Swizzle-Switch*, provides an energy- and area-efficient switching element that improves the scalability of crossbars to a high radices. The multicast ability and integrated arbitration of the *Swizzle-Switch* makes is shown to operate at frequencies up to 1.5GHz for 128-bit, radix-

64 crossbars and have the ability to implement many arbitration policies such as Least-Recently Granted (LRG) and Round-Robin (RR). Results show that *Swizzle-Switch*'s LRG arbitration policy reduces the worst-case request access latency by 1.83 and 2.03 on average over round robin and random arbitration schemes, respectively.

After the *Swizzle-Switch* is scaled to 32nm, Chapter IV uses multiple *Swizzle-Switches* are used to create the *Swizzle-Switch Network*: a flat, cache-coherent crossbar topology supporting many-core systems of up to 64 cores. The *Swizzle-Switch* is also used as a high-radix building block that can enable high-radix topologies such as the Flattened Butterfly (FBFly). The evaluation in Chapter IV shows that the *Swizzle-Switch Network* scales favorably over a traditional Network-on-Chip (NoC) topology (Mesh) for systems of up to 64 cores. On average, the FBFly is 15% faster, has a 1.76× smaller L1 on-chip average miss latency, a 2.5× reduction in miss latency standard deviation, and a 10% energy savings over the Mesh. The SSN improves system performance by 21%, with a 2.2× smaller L1 on-chip average miss latency, and a 3.0× reduction in miss latency standard deviation all while providing a 25% energy savings compared to the Mesh. These improvements show that the FBFly and SSN facilitate easier programmability and quality of service guarantees on many-core systems.

The impact of 3D stacking technology has on many-core scalability is studied in Chapter V. The study finds that 3D integration technology can help crossbar and bus interconnects scale past their traditional limitations. In particular, a 3D-*Swizzle-Switch Network* architecture is presented and shown to operate at 50% and 80% higher frequencies when using 2-layer and 4-layer stacks respectively. The 3D-optimized SSN is able to leverage these frequency gains to achieve 15-28% speedup on memory-intensive benchmarks and 10% average speedup across all benchmarks. Additionally, the bus-based *XPoint* architecture uses 3D to optimize both contention and latency over conventional bus interconnects. As such, a 64-core *XPoint-3D* system obtains a 49× speedup over the baseline uniprocessor system.

Challenges in interconnects for future many-core systems remain as on-chip core counts continues to rise past 100. Interconnects will need to be further optimized for heterogenous systems while simultaneously respect area and power constraints. This thesis suggests that for systems of up to 64 cores, a flat-crossbar interconnect like the *Swizzle-Switch Network* can be used to optimize performance, power, and quality-of-service issues. For systems past 64 cores, high-radix topologies like the Flattened Butterfly can use the *Swizzle-Switch* to continue to scale performance. Additionally, this thesis shows that 3D architectures can be utilized to scale bus architectures to 64 cores and are promising alternatives for scaling future many-core systems past 64 cores (*e.g*. high-radix 3D-NoC topologies). Future research can take advantage of the feasible high-radix interconnects shown in this work for future flat, hierarchical, and 3D interconnect architectures that can support many-core scaling past 100 cores.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] T. W. Ainsworth and T. M. Pinkston. Characterizing the cell eib on-chip network. *IEEE Micro*, 2007.

[2] A. Alameldeen and D. Wood. Variability in architectural simulations of multi-threaded workloads. In *High-Performance Computer Architecture, 2003. HPCA-9 2003.*, pages 7 –18, feb. 2003.

[3] J. Andrews and N. Baker. Xbox 360 system architecture. *IEEE Micro*, 2006.

[4] ARM Ltd. Cortex-A5 Processor. *ARM Databrief*, 2010.

[5] Arm Ltd. Amba open specifications: Arm. `http://www.arm.com/products/system-ip/amba/amba-open-specifications.php`, March 2012.

[6] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz. Energy-performance tradeoffs in processor architecture and circuit design: a marginal cost analysis. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 26–36, New York, NY, USA, 2010. ACM.

[7] J. Balfour and W. J. Dally. Design tradeoffs for tiled cmp on-chip networks. In *ICS-20*, 2006.

[8] N. Barrow-Williams, C. Fensch, and S. Moore. A communication characterisation of splash-2 and parsec. In *IISWC*, 2009.

[9] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *Computer Architecture News (CAN)*, Jun 2011.

[10] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 2006.

[11] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 2006.

[12] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die stacking (3d) microarchitecture. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 469–479, Washington, DC, USA, 2006. IEEE Computer Society.

[13] L. Bononi and N. Concer. Simulation and analysis of network-on-chip architectures: Ring, spidergon and 2d mesh. In *DATE*, 2006.

[14] L. Bononi, N. Concer, M. Grammatikakis, M. Coppola, and R. Locatelli. Noc topologies exploration based on mapping and simulation models. In *EUROMICRO-10*, 2007.

[15] S. Borkar. Networks for multi-core chips: A contrarian view. *Special Session at ISLPED*, 2007.

[16] J. Burns and J.-L. Gaudiot. Smt layout overhead and scalability. *Parallel and Distributed Systems, IEEE Transactions on*, 13(2):142 –155, feb 2002.

[17] F. J. Cazorla, A. Ramirez, M. Valero, and E. Fernandez. Dynamically controlled resource allocation in smt processors. In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 37, pages 171–182, Washington, DC, USA, 2004. IEEE Computer Society.

[18] R. S. Chappell, J. Stark, S. P. Kim, S. K. Reinhardt, and Y. N. Patt. Simultaneous subordinate microthreading (ssmt). In *Proceedings of the 26th annual international symposium on Computer architecture*, ISCA '99, pages 186–195, Washington, DC, USA, 1999. IEEE Computer Society.

[19] C.-H. Chen, G.-W. Lee, J.-D. Huang, and J.-Y. Jou. A real-time and bandwidth guaranteed arbitration algorithm for soc bus communication. In *Design Automation, 2006. Asia and South Pacific Conference on*, page 6 pp., jan. 2006.

[20] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. B. Carter. Interconnect-aware coherence protocols for chip multiprocessors. In *ISCA-33*, 2006.

[21] G.-M. Chiu. The odd-even turn model for adaptive routing. *Parallel and Distributed Systems, IEEE Transactions on*, 11(7):729 –738, jul 2000.

[22] M. Coppola, M. D. Grammatikakis, R. Locatelli, G. Maruccia, and L. Pieralisi. *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2008.

[23] D. Culler, J. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1st edition, 1998. The Morgan Kaufmann Series in Computer Architecture and Design.

[24] W. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *Parallel and Distributed Systems, IEEE Transactions on*, 4(4):466 –475, apr 1993.

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

[26] W. J. Dally. Virtual-channel flow control. *Parallel and Distributed Systems, IEEE Transactions on*, 3(2):194 –205, mar 1992.

[27] W. J. Dally and C. L. Seitz. The torus routing chip. *Distributed Computing*, 1:187–196, 1986. 10.1007/BF01660031.

[28] R. Das, S. Eachempati, A. K. Mishra, V. Narayanan, and C. R. Das. Design and evaluation of a hierarchical on-chip interconnect for next-generation cmps. In *HPCA-15*, 2009.

[29] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. Leo Rideovt, E. Bassous, and A. R. Leblanc. Design of ion-implanted mosfet's with very small physical dimensions. *Journal of Solid-State Circuits, IEEE*, SC-9:256–268, october 1974.

[30] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *Parallel and Distributed Systems, IEEE Transactions on*, 4(12):1320 –1331, dec 1993.

[31] J. Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *Parallel and Distributed Systems, IEEE Transactions on*, 6(10):1055 –1067, oct 1995.

[32] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[33] D. Fick, R. Dreslinski, et al. Centip3de: A 3930 dmips/w configurable near-threshold 3d stacked system with 64 arm cortex-m3 cores. *To appear in IEEE International Solid-State Circuits Conference, San Francisco, CA, 2012*, 2012.

[34] M. Galles. Spider: A high-speed network interconnect. *IEEE Micro*, 17:34–39, January 1997.

[35] M. Galles and E. Williams. Performance optimizations, implementation, and verification of the sgi challenge multiprocessor. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, 1994.

[36] M. Ghoneima and Y. Ismail. Optimum positioning of interleaved repeaters in bidirectional buses. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(3):461 – 469, march 2005.

[37] C. Glass and L. Ni. The turn model for adaptive routing. In *Computer Architecture, 1992. Proceedings., The 19th Annual International Symposium on*, pages 278 –287, 1992.

[38] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu. Express cube topologies for on-chip interconnects. In *HPCA-15*, 2009.

[39] B. Grot, S. W. Keckler, and O. Mutlu. Preemptive virtual clock: a flexible, efficient, and cost-effective qos scheme for networks-on-chip. In *MICRO 42*, pages 268–279, New York, NY, USA, 2009. ACM.

[40] M. Gschwind, H. Hofstee, B. Flachs, M. Hopkin, Y. Watanabe, and T. Yamazaki. Synergistic processing in cell's multicore architecture. *Micro, IEEE*, 2006.

[41] S. Gupta, M. Hibert, S. Hong, and R. Patti. Techniques for producing 3d ics with high-density interconnect. *White Paper*, 2004.

[42] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Reactive nuca: near-optimal block placement and replication in distributed caches. In *Proceedings of the*

36th annual international symposium on Computer architecture, ISCA '09, pages 184–195, New York, NY, USA, 2009. ACM.

[43] R. Haring. The ibm blue gene/q compute chip+simd floating-point unit. In *HotChips 23: A Symposium on High-Performance Chips*, 2011.

[44] G. Hinton, R. Riches, C. Jasper, and K. Lai. A register scoreboarding mechanism. In *Solid-State Circuits Conference, 1988. Digest of Technical Papers. ISSCC. 1988 IEEE International*, pages 270 –271, feb. 1988.

[45] J. Howard, S. Dighe, et al. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *ISSCC 2010*, pages 108 –109, feb. 2010.

[46] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular noc architectures. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(4):551 – 562, april 2005.

[47] Y. J. Hwang, J. H. Lee, and T. H. Han. 3d network-on-chip system communication using minimum number of tsvs. In *ICT Convergence (ICTC), 2011 International Conference on*, pages 517 –522, sept. 2011.

[48] Intel Corp. Intel introduces the pentium 4 processor. `http://web.archive.org/web/20070403032914/http://www.intel.com/pressroom/archive/releases/dp112000.htm`, November 2000.

[49] Intel Corp. Intel atom processor for nettop platforms. *Intel Product Brief*, Novemeber 2008.

[50] Intel Corp. Intel core i7-2600 processor (8m cache, 3.40 ghz). `http://ark.intel.com/products/52213`, March 2012.

[51] ISSCC. International solid-state circuits conference (isscc) 2011 trends report. `http://isscc.org/doc/2011/2011_Trends.pdf`, February 2011.

[52] N. Jerger and L.-S. Peh. *On-Chip Networks*. Morgan ClayPool Publishers, 1st edition, 2009.

[53] N. E. Jerger, L.-S. Peh, and M. Lipasti. Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In *Proc. of the 35th Annl. Intnl. Symposium on Computer Architecture*, ISCA '08, pages 229–240, 2008.

[54] T. Johnson and U. Nawathe. An 8-core, 64-thread, 64-bit power efficient sparc soc (niagara2). In *ISPD*, 2007.

[55] C. Keltcher, K. McGrath, A. Ahmed, and P. Conway. The amd opteron processor for multiprocessor servers. *Micro, IEEE*, 23(2):66 – 76, march-april 2003.

[56] P. Kermani and L. Kleinrock. Virtual cut-through: a new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[57] T. Kgil, S. D'Souza, A. G. Saidi, N. L. Binkert, R. G. Dreslinski, T. N. Mudge, S. K. Reinhardt, and K. Flautner. Picoserver: using 3d stacking technology to enable a compact energy efficient chip multiprocessor. In *ASPLOS*, pages 117–128, 2006.

[58] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *ASPLOS-X*, 2002.

[59] D. H. Kim, K. Athikulwongse, et al. 3d-maps: 3d massively parallel processor with stacked memory. *To appear in IEEE International Solid-State Circuits Conference, San Francisco, CA, 2012*, 2012.

[60] D. H. Kim, S. Mukhopadhyay, and S. K. Lim. Tsv-aware interconnect length and power prediction for 3d stacked ics. In *Interconnect Technology Conference, 2009. IITC 2009. IEEE International*, pages 26 –28, june 2009.

[61] J. Kim, J. Balfour, and W. Dally. Flattened butterfly topology for on-chip networks. In *MICRO-40*, 2007.

[62] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta. Microarchitecture of a high-radix router. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 420–431, Washington, DC, USA, 2005. IEEE Computer Society.

[63] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: a 32-way multithreaded sparc processor. *Micro, IEEE*, 25(2):21 – 29, march-april 2005.

[64] G. Kornaros. Bcb: A buffered crossbar switch fabric utilizing shared memory. In *EUROMICRO-9*, 2006.

[65] K. Kuhn. Cmos transistor scaling past 32nm and implications on variation. In *Advanced Semiconductor Manufacturing Conference (ASMC), 2010 IEEE/SEMI*, pages 241 –246, july 2010.

[66] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *ISCA-32*, 2005.

[67] K. Lahiri, A. Raghunathan, and G. Lakshminarayana. Lotterybus: a new high-performance communication architecture for system-on-chip designs. In *Design Automation Conference, 2001. Proceedings*, pages 15 – 20, 2001.

[68] K. Lahiri, A. Raghunathan, and G. Lakshminarayana. The lotterybus on-chip communication architecture. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(6):596 –608, june 2006.

[69] K. Lahiri, A. Raghunathan, G. Lakshminarayana, and S. Dey. Communication architecture tuners: a methodology for the design of high-performance communication architectures for system-on-chips. In *Design Automation Conference, 2000. Proceedings 2000. 37th*, pages 513 –518, 2000.

[70] K. Lahiri, A. Raghunathan, G. Lakshminarayana, and S. Dey. Design of high-performance system-on-chips using communication architecture tuners. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(5):620 – 636, may 2004.

[71] J. W. Lee, M. C. Ng, and K. Asanovic. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. In *Proc. of the 35th Annl. Intl. Symp. on Computer Architecture*, ISCA '08, pages 89–100, 2008.

[72] L. Li, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and I. Kadayif. Ccc: crossbar connected caches for reducing energy consumption of on-chip multiprocessors. In *EUROMICRO*, 2003.

[73] B.-C. Lin, G.-W. Lee, J.-D. Huang, and J.-Y. Jou. A precise bandwidth control arbitration algorithm for hard real-time soc buses. In *Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific*, pages 165 –170, jan. 2007.

[74] P. Lindner, V. Dragoi, T. Glinsner, C. Schaefer, and R. Islam. 3d interconnect through aligned wafer level bonding. In *Electronic Components and Technology Conference, 2002. Proceedings. 52nd*, pages 1439 – 1443, 2002.

[75] C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari. Bridging the processor-memory performance gap with 3d ic technology. *Design Test of Computers, IEEE*, 22(6):556 – 564, nov.-dec. 2005.

[76] A. Ltd. http://www.arm.com/products/processors/cortex-a/cortex-a9.php, 2011.

[77] M. R. Marty and M. D. Hill. Virtual hierarchies to support server consolidation. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 46–56, 2007.

[78] D. May. Xmos xs1 architectures. *XMOS Ltd.*, July 2008.

[79] N. McKeown. The islip scheduling algorithm for input-queued switches. *Networking, IEEE/ACM Transactions on*, 7(2):188 –201, apr 1999.

[80] C. McNairy and D. Soltis. Itanium 2 processor microarchitecture. *Micro, IEEE*, 23(2):44 – 55, march-april 2003.

[81] S. Mirapuri, M. Woodacre, and N. Vasseghi. The mips r4000 processor. *Micro, IEEE*, 12(2):10 –22, april 1992.

[82] G. E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *Solid-State Circuits Newsletter, IEEE*, 20(3):33 –35, sept. 2006.

[83] G. E. Moore. Lithography and the future of moore's law, copyright 1995 ieee. reprinted with permission. proc. spie vol. 2437, pp. 2. *Solid-State Circuits Newsletter, IEEE*, 20(3):37 –42, sept. 2006.

[84] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The alpha 21364 network architecture. *Micro, IEEE*, 22(1):26 –35, jan/feb 2002.

[85] R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 188 – 197, june 2004.

[86] T. Nesson and S. L. Johnsson. Romm routing on mesh and torus networks. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, SPAA '95, pages 275–287, New York, NY, USA, 1995. ACM.

[87] D. W. Oehmke, N. L. Binkert, T. Mudge, and S. K. Reinhardt. How to fake 1000 registers. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 38, pages 7–18, Washington, DC, USA, 2005. IEEE Computer Society.

[88] S. Pasricha and N. Dutt. *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann, 1st edition, 2008.

[89] G. Passas, M. Katevenis, and D. Pnevmatikatos. A 128 x 128 x 24gb/s crossbar interconnecting 128 tiles in a single hop and occupying 6% of their area. In *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS, 2010.

[90] G. Passas, M. Katevenis, and D. Pnevmatikatos. Vlsi micro-architectures for high-radix crossbar schedulers. In *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS, 2011.

[91] L.-S. Peh and W. J. Dally. A delay model and speculative architecture for pipelined routers. In *Proc. of the 7th Intl. Symp. on High-Performance Computer Architecture*, HPCA '01, pages 255–, 2001.

[92] K. Puttaswamy and G. Loh. The impact of 3-dimensional integration on the design of arithmetic units. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, page 4 pp., may 2006.

[93] S. E. Raasch and S. K. Reinhardt. The impact of resource partitioning on smt processors. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, PACT '03, pages 15–, Washington, DC, USA, 2003. IEEE Computer Society.

[94] E. Ranney. Alr hopes to beat completion with fall release of 386 line. *InfoWorld*, page 5, September 1986.

[95] S. Rusu. Keynote:trends and challenges in high-performance microprocessor design. presented at electronic design processes 2004. monterey beach, ca. `www.eda.org/edps/edp04/submissions/presentationRusu.pdf`, April 2004.

[96] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis. An analysis of on-chip interconnection networks for large-scale chip multiprocessors. *ACM TACO*, 2010.

[97] S. Satpathy, R. Das, K. Sewell, R. Dreslinski, D. Sylvester, T. Mudge, and D. Blaauw. High radix self-arbitrating switch fabric with multiple arbitration schemes and quality of service. In *Design Automation Conference (DAC) 2012*, 2012.

[98] S. Satpathy, R. Dreslinski, T. Ou, D. Sylvester, T. Mudge, and D. Blaauw. Swift: A 2.1tb/s 3232 self-arbitrating manycore interconnect fabric. In *VLSI Circuits (VLSIC), 2011 IEEE Symposium on*, june 2011.

[99] S. Satpathy, Z. Foo, B. Giridhar, R. Dreslinski, D. Sylvester, T. Mudge, and D. Blaauw. A 1.07 tbit/s 128x128 swizzle network for simd processors. In *VLSIC*, 2010.

[100] S. Satpathy, K. Sewell, T. Manville, R. Dreslinski, D. Sylvester, T. Mudge, and D. Blaauw. A 4.5tb/s 3.4tb/s/w 64x64 switch fabric with self-updating least recently granted priority and quality of service arbitration in 45nm cmos. In *IEEE International Solid-State Circuits Conference (ISSCC), February 2012*, 2012.

[101] J. Schanin. The design and development of a very high speed system bus: The encore multimax nanobus. In *ACM Fall Joint Computer Conference*, 1986.

[102] S. Scott, D. Abts, J. Kim, and W. J. Dally. The blackwidow high-radix clos network. In *Proceedings of the 33rd annual international symposium on Computer Architecture*, ISCA '06, pages 16–28, Washington, DC, USA, 2006. IEEE Computer Society.

[103] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM Transactions on Graphics*, 2008.

[104] 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 *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, pages 29 – 40, feb. 2002.

[105] K. Sewell, T. Mudge, and S. K. Reinhardt. Extreme virtual pipelining (xvp): Moving towards scalable multithreaded processors. In *Wild and Crazy Ideas held in conjunction with 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2009.

[106] A. Singh, W. Dally, A. Gupta, and B. Towles. Goal: a load-balanced adaptive routing algorithm for torus networks. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pages 194 – 205, june 2003.

[107] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pages 2 – 13, june 2003.

[108] Soncis Inc. Sonics unetwork technical overview. `http://www.sonicsinc.com`, January 2002.

[109] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. `http://www.gotw.ca/publications/concurrency-ddj.htm`, August 2009.

[110] R. W. Technologies. Alpha ev8 (part 2): Simultaneous multi-threat. `http://www.realworldtech.com/page.cfm?ArticleID=RWT122600000000`, December 2000.

[111] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy. Power4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5 –25, jan. 2002.

[112] R. M. Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal of Research and Development*, 11(1):25 –33, jan. 1967.

[113] Y.-F. Tsai, Y. Xie, N. Vijaykrishnan, and M. Irwin. Three-dimensional cache design exploration using 3dcacti. In *Computer Design: VLSI in Computers and Processors, 2005. ICCD 2005. Proceedings. 2005 IEEE International Conference on*, pages 519 – 524, oct. 2005.

[114] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm. Exploiting choice: instruction fetch and issue on an implementable simultaneous

multithreading processor. In *Proceedings of the 23rd annual international symposium on Computer architecture*, ISCA '96, pages 191–202, New York, NY, USA, 1996. ACM.

[115] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: maximizing on-chip parallelism. In *Proceedings of the 22nd annual international symposium on Computer architecture*, ISCA '95, pages 392–403, New York, NY, USA, 1995. ACM.

[116] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, STOC '81, pages 263–277, New York, NY, USA, 1981. ACM.

[117] J. C. Villanueva, J. Flich, J. Duato, H. Eberle, N. Gura, and W. Olesinski. A performance evaluation of 2d-mesh, ring, and crossbar interconnects for chip multiprocessors. In *NoCArc-2*, 2009.

[118] H.-S. Wang, L.-S. Peh, and S. Malik. A power model for routers: Modeling alpha 21364 and infiniband routers. In *HOTI-10*, 2002.

[119] Wasson, Scott. Intel's pentium 4 prescott processor. `http://techreport.com/articles.x/6213/1`, February 2004.

[120] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. Brown, and A. Agarwal. On-chip interconnection architecture of the tile processor. *Micro, IEEE*, 27(5):15–31, sept.-oct. 2007.

[121] D. Winsor and T. N. Mudge. Crosspoint cache architectures. In *In International Symposium on Computer Architecture*, pages 266–269, 1987.

[122] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *ISCA-22*, 1995.

[123] Xbit Labs. Intel pentium 4 3.06ghz cpu with hyper-threading technology: Killing two birds with a stone... `http://www.xbitlabs.com/articles/cpu/display/pentium4-3066.html`, November 2002.

[124] K. Yeager. The mips r10000 superscalar microprocessor. *Micro, IEEE*, 16(2):28–41, apr 1996.

[125] Y. Zhang. Architecture and performance comparison of a statistic-based lottery arbiter for shared bus on chip. In *Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific*, volume 2, pages 1313 – 1316 Vol. 2, jan. 2005.

[126] Y. Zhang and M. J. Irwin. Power and performance comparison of crossbars and buses as on-chip interconnect structures. In *ASILOMAR-33*, 1999.

[127] Y. P. Zhang, T. Jeong, F. Chen, H. Wu, R. Nitzsche, and G. Gao. A study of the on-chip interconnection network for the ibm cyclops64 multi-core architecture. In *IPDPS-20*, 2006.