WIRE AWARE CACHE ARCHITECTURE

by

Naveen Muralimanohar

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

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

December 2009

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Naveen Muralimanohar

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

	Chair: Rajeev Balasubramonian
	Al Davis
	John Carter
	Erik Brunvand
	Shubu Mukherjee

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

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

Date

Rajeev Balasubramonian
Chair: Supervisory Committee

Approved for the Major Department

Martin Berzins
Chair/Director

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School

ABSTRACT

Technology scaling has resulted in a steady increase in transistor speed. However, unlike transistors, global wires that span across the chip show a reverse trend of getting slower with shrinking process. Modern processors are severely constrained by wire delay and the widening gap between transistors and wires will only exacerbate the problem. Following the traditional design approach of adopting a single design point for all global wires will be suboptimal in terms of both power and performance. VLSI techniques allow several wire implementations with varying latency, power, and bandwidth properties. The dissertation advocates exposing wire properties to architects and demonstrates that prudent management of wires at the microarchitectural level can lead to significant improvement in power and delay characteristics of future communication bound processors. A heterogeneous interconnect (composed of wires with different latency, bandwidth, and power characteristics) is proposed that leverages varying latency and bandwidth needs of on-chip global messages to alleviate interconnect overhead.

The effect of interconnect is more pronounced in large on-chip structures. Since on-chip caches occupy more than 50% of the microprocessor real estate and up to 80% of their access time is attributed to the wire delay, the dissertation focuses on the prudent management of wires for cache accesses. This work demonstrates that a cache design that pays special attention to network parameters and wire properties can be very different from traditional cache models and performs significantly better. The dissertation studies every aspect of communication happening in on-chip caches and proposes techniques that leverage heterogeneous interconnect to reduce their overhead. Finally, in an effort to make the interconnect model explicit to architects and impact future research, a state-of-the-art wire and router modeling is incorporated into a publicly available CACTI tool along with novel

methodologies to accurately estimate the delay, power, and area of large caches with uniform and nonuniform access times.

To My Parents

CONTENTS

ABSTRACT . . . . . **iv**

LIST OF FIGURES . . . . . **x**

LIST OF TABLES . . . . . **xii**

ACKNOWLEDGMENTS . . . . . **xiii**

CHAPTERS

1. INTRODUCTION **1**

1.1 Emerging Challenges 1
1.2 Wire Aware Design 3
1.3 Thesis Statement 4
1.4 Dissertation Overview 6

2. DESIGN SPACE OF ON-CHIP WIRES **8**

2.1 Wire Implementations with Varying Characteristics 8
2.1.1 Wire Width and Spacing 9
2.1.2 Repeater Size and Spacing 9
2.1.3 Low-swing Wires 11
2.1.4 Transmission Lines 12
2.2 Heterogeneous Interconnects 13
2.3 Analytical Models 14
2.3.1 Global Wires 14
2.3.2 Low-swing Wires 16
2.4 Summary 22

3. INTERCONNECT DESIGN FOR LARGE CACHES **24**

3.1 Need for an Enhanced Cache Modeling Tool 25
3.2 CACTI 6.0 27
3.2.1 Interconnect Modeling for UCA Caches 29
3.2.2 NUCA Modeling 33
3.2.3 Router Models 38
3.2.4 Improvement in Trade-Off Analysis 42
3.3 Case Study 43
3.4 Validation 46
3.5 Related Work 50

3.6 Summary . . . 52

4. NOVEL CACHE PIPELINES . . . 54

4.1 Leveraging Interconnect Choices for Performance Optimizations . . . 54
4.1.1 Early Look-Up . . . 54
4.1.2 Aggressive Look-Up . . . 57
4.1.3 Hybrid Network . . . 59
4.2 Results . . . 60
4.2.1 Methodology . . . 60
4.2.2 IPC Analysis . . . 62
4.3 Related Work . . . 67
4.4 Summary . . . 69

5. HETEROGENEOUS INTERCONNECTS FOR COHERENCE TRANSACTIONS . . . 71

5.1 Coherence Overhead . . . 71
5.2 Optimizing Coherence Traffic . . . 72
5.2.1 Protocol-Dependent Techniques . . . 72
5.2.2 Protocol-independent Techniques . . . 77
5.3 Implementation Complexity . . . 78
5.3.1 Overhead in Heterogeneous Interconnect Implementation . . . 78
5.3.2 Overhead in Decision Process . . . 80
5.3.3 Overhead in Cache Coherence Protocols . . . 80
5.4 Methodology . . . 81
5.4.1 Simulator . . . 81
5.4.2 Interconnect Configuration . . . 82
5.5 Results . . . 85
5.6 Sensitivity Analysis . . . 89
5.6.1 Out-of-order/In-order Processors . . . 89
5.6.2 Link Bandwidth . . . 90
5.6.3 Routing Algorithm . . . 90
5.6.4 Network Topology . . . 91
5.7 Related Work . . . 92
5.8 Summary . . . 93

6. WIRE MANAGEMENT IN A CLUSTERED ARCHITECTURE 95

6.1 Clustered Architecture . . . 95
6.2 The Baseline Partitioned Architecture . . . 97
6.3 Exploiting Heterogeneous Interconnects . . . 99
6.3.1 Accelerating Cache Access . . . 99
6.3.2 Narrow Bit-Width Operands . . . 102
6.3.3 Exploiting PW-Wires . . . 102
6.4 Results . . . 103

6.4.1 Methodology 103
6.4.2 Latency and Energy Estimates 105
6.4.3 Behavior of L-Wires 106
6.4.4 Heterogeneous Interconnect Choices 108
6.5 Related Work 112
6.6 Summary 113

7. CONCLUSIONS AND FUTURE WORK 114

7.1 Impact of the Dissertation 117
7.2 Future Work 117

REFERENCES 120

LIST OF FIGURES

2.1 Examples of different wire implementations. Energy optimized wires have fewer and smaller repeaters, while bandwidth optimized wires have narrow widths and spacing. 10

2.2 Effect of repeater spacing/sizing on wire delay. 11

2.3 Contours for 2% delay penalty 12

2.4 Interconnect segment 15

2.5 Low-swing transmitter (actual transmitter has two such circuits to feed the differential wires) 18

2.6 Sense-amplifier circuit 23

3.1 Contribution of H-tree network to overall cache delay and power 27

3.2 Logical organization of the cache (from CACTI 3.0 [108]) 28

3.3 Physical organization of the cache (from CACTI 3.0 [108]) 29

3.4 Delay characteristics of different wires 30

3.5 Energy characteristics of different wires 30

3.6 Design space exploration with global wires 31

3.7 Design space exploration with full-swing global wires (red, bottom region), wires with 30% delay penalty (yellow, middle region), and differential low-swing wires (blue, top region) 32

3.8 8-bank data array with a differential low-swing broadcast bus 33

3.9 Total network contention value/access for CMPs with different NUCA organizations 37

3.10 NUCA design space exploration 38

3.11 Router architecture 40

3.12 Router pipeline [42] 40

3.13 Access frequency for a 32MB cache. The y-coordinate of a point in the curve corresponds to the percentage of accesses that can be satisfied with x KB of cache 46

3.14 Low-swing model delay verification 48

3.15 Low-swing model energy verification 49

3.16 Distributed wordline model verification 50

3.17 Distributed bitline model verification 51
4.1 Hybrid network topology for a uniprocessor. 60
4.2 Normalized IPCs of SPEC2000 benchmarks for different L2 cache configurations (B-wires implemented on the 8X metal plane). 64
4.3 IPCs of SPEC2000 benchmarks for different L2 cache configurations (B-wires implemented on the 8X metal plane). 65
4.4 IPC improvement of different cache configurations in an eight core CMP. Benchmark compositions: "Mix" - ammp, applu, lucas, bzip2, crafty, mgrid, equake, gcc; "All sensitive" - ammp, apsi, art, bzip2, crafty, eon, equake, gcc; "Half L2 and Half Non-L2 sensitive" - ammp, applu, lucas, bzip2, crafty, mgrid, mesa, gcc; "Memory intensive" - applu, fma3d, art, swim, lucas, equake, gap, vpr. 67
4.5 Performance for uniprocessor with 4X wires. 68
5.1 Read exclusive request for a shared block in MESI protocol 74
5.2 Interconnect model used for coherence transactions in a sixteen-core CMP. 83
5.3 Speedup of heterogeneous interconnect 86
5.4 Distribution of messages on the heterogeneous network. B-Wire transfers are classified as Request and Data. 87
5.5 Distribution of L-message transfers across different proposals. 88
5.6 Improvement in link energy and ED^2. 89
5.7 Speedup of heterogeneous interconnect when driven by OoO cores (Opal and Ruby) 91
5.8 2D Torus topology 92
5.9 Heterogeneous interconnect speedup 93
6.1 Clustered architecture with heterogeneous interconnect. (a) A partitioned architecture model with 4 clusters and a heterogeneous interconnect comprised of *B*-, *L*-, and *PW-Wires*. (b) A 16-cluster system with a hierarchical interconnect. Sets of four clusters are connected with a crossbar and the crossbars are connected in a ring topology. . . 99
6.2 IPCs for the baseline 4-cluster partitioned architecture employing one layer of *B-Wires* and for a partitioned architecture employing one layer of *B-Wires* and one layer of *L-Wires*. The *L-Wires* transmit narrow bit-width data, branch mispredict signals, and LS bits of load/store addresses. 107

LIST OF TABLES

2.1 Sense-amplifier delay and energy values for different process technologies 23

3.1 Benchmark sets 35

3.2 Simplescalar simulator parameters 36

3.3 Energy consumed (in J) by arbiters, buffers, and crossbars for various router configurations at 32nm technology (flit size of 128 bits). 42

3.4 Delay and energy values of different components for a 5mm low-swing wire. 49

4.1 Delay and area characteristics of different wire implementations at 65 nm technology. 55

4.2 Simplescalar simulator parameters. 61

4.3 Summary of different models simulated. Global 8X wires are assumed for the inter-bank links. "A" and "D" denote the address and data networks, respectively. 62

4.4 Access latencies for different cache configurations. The operating frequency is 5GHz and the message transfers are assumed to happen on 8x wires. 66

5.1 Power characteristics of different wire implementations. For calculating the power/length, activity factor α (described in Table 5.3) is assumed to be 0.15. The above latch spacing values are for a 5GHz network. 79

5.2 System configuration. 82

5.3 Area, delay, and power characteristics of different wire implementations. 84

6.1 Simplescalar simulator parameters. 104

6.2 Wire delay and relative energy parameters for each RC-based wire. . . 106

6.3 Heterogeneous interconnect energy and performance for 4-cluster systems. All values (except IPC) are normalized with respect to $Model-I$. ED^2 is computed by multiplying total processor energy by square of executed cycles. 10% and 20% refer to the contribution of interconnect energy to total processor energy in $Model-I$. 109

6.4 Heterogeneous interconnect energy and performance for 16-cluster systems where interconnect energy contributes 20% of total processor energy in $Model-I$. All values (except IPC) are normalized with respect to $Model-I$. 111

ACKNOWLEDGMENTS

During the course of my graduate studies, I have had the opportunity to work with some great minds and it is a pleasure to thank those who helped me get to this point. First and foremost, I owe my heartfelt gratitude to my adviser, Dr. Rajeev Balasubramonian, for providing me with this opportunity and always being there for me throughout the graduate studies. His attention to detail, high standards, and clarity in thinking are qualities I would strive to inherit and use throughout my career. I cannot thank him enough for his unflinching encouragement and patience, which helped me cross many barriers and made my graduate studies a joyful experience.

My special thanks to Dr. Al Davis who not only inspired me towards Computer Architecture but also has been a great source of support for many electrical problems. I would like to thank the other members of my committee, Dr. John Carter, Dr. Shubu Mukherjee, and Dr. Erik Brunvand, for their constructive feedback, which helped improve my dissertation. I am indebted to Dr. Norm Jouppi, my mentor at HP Labs, for his unwavering support and the freedom he gave me during my internship. I am also thankful to Dr. Partha Ranganathan for his tips on numerous topics including job hunt, career, industry research, etc.

I would like to thank my friends and the co-authors, especially Karthik Ramani, for all the great memories and for the sleepless nights we were working together before deadlines - Niti, Anirudh, Liqun, Manu, and Seth - with whom I had numerous thought provoking discussions. My sincere thanks to all the architecture group members at Utah who provided me valuable feedback on my work and presentations. I am grateful to my roomates, Sachin Goyal and Piyush Rai, for their company and for being extremely cooperative during deadlines.

Above all, I am extremely fortunate to have great family members who encouraged and helped me in every possible way. Without their moral support, none of this would have been possible.

CHAPTER 1

INTRODUCTION

1.1 Emerging Challenges

For the past three decades, there has been a steady increase in processor performance due to shrinking of feature size and microarchitectural innovations. A recent analysis shows that in a time frame of 10 years, processor performance has improved by a factor of 55 [46] because of rising clock frequency and advancements in microarchitecture. While industry projections continue to show a healthy road map for technological advancements, the realization of a similar performance boost in the next 10 years is being severely challenged by technology constraints and architecture complexity. Continued scaling of transistors has posed a number of new challenges to the architecture community.

- **Power Budget**

  Power consumption in a processor is a function of transistor density (a transistor load also includes capacitive load of connecting wires) and switching frequency. With a modern processor's transistor budget exceeding the billion count, we have reached a point where we cannot afford to operate all transistors in a chip at the same time. In the past, increased transistor budget with every process generation is also accompanied by a scaled down operating voltage. Since power consumption has a quadratic dependence on voltage, lowering operating voltage significantly helps reduce power consumption. However, with the growing concern of process variation and other reliability issues (discussed in the next subsection) very low operating voltage is no longer an option in deep submicron technologies. As a result, architects designing future processors will need to work with a strict power and tem-

perature budget. Novel microarchitectural techniques are required to eke out performance with a limited power budget.

- **Reliability**

  Shrinking transistor dimensions coupled with high clock speed and low operating voltage has introduced a number of reliability issues such as parameter variation and soft errors. With the continued scaling of transistors, we have reached a point where it is extremely difficult to precisely control the dimensions of a transistor. Given the sensitivity of transistor properties to its dimensions, process variation can significantly affect the latency and leakage characteristics of transistors. High operating frequencies of modern processors further complicate the problem by providing very little time for transistors to reach the stable state. Radiation induced soft errors [89] and V_{th} (transistor threshold voltage) degradation [29] are other emerging problems that constrain scaling down operating voltage. As a net result of these new problems, future microarchitecture design can no longer focus singularly on performance. Architects should also innovate techniques to improve processor robustness to tolerate process parameter variation and soft errors.

- **Wire Delay**

  Global wires are another major show stopper in modern processors. While arithmetic computation continues to consume fewer pico-seconds and die area with every generation, the cost of on-chip communication continues to increase [84]. Traditional electrical interconnects are viewed as a critical limiting factor, with regard to latency, bandwidth, and power. The ITRS road-map projects that global wire speeds will degrade substantially at smaller technologies and a signal on a global wire can consume over 12 ns (60 cycles at a 5 GHz frequency) to traverse 20 mm at 32 nm technology [7]. In some Intel chips, half the total dynamic power is attributed to interconnects [79]. Hence, a careful analysis and a better understanding of interconnects are necessary to alleviate the deleterious effects of wire scaling.

1.2 Wire Aware Design

Most architecture proposals in the recent past focus directly or indirectly on solving these new challenges. Of all the problems described above, the problems associated with wire scaling are particularly compelling for two reasons. First, all existing architectural techniques aimed at reducing wire delay or wire power are indirect. For example, microarchitectural techniques such as prefetching, run-ahead execution, speculative coherence, or speculation in other forms try to hide wire delays rather than reduce them. Similarly, a reduction in wire power is achieved by avoiding communication instead of directly reducing the interconnect power. Direct wire level optimizations are still limited to circuit designers. Second, traditional design approaches employ a single wire choice optimized for the average case for all global wires in a processor. In the past, this was not a big issue since both processor performance and power dissipation are typically determined by the characteristics of transistors. However, with the current trend of processors becoming more and more communication bound, the choice of interconnect parameters will have a dramatic impact on a processor's power and performance characteristics.

Focusing on interconnect design can also help alleviate other critical problems, such as power consumption and processor reliability discussed in the previous section. For example, though a low operating voltage is not a favorable option for transistors, recent studies [53, 130] show that it is a powerful technique to reduce interconnect power in modern processors. With on-chip wires accounting for 30-50% of the processor power [79, 122], reducing interconnect power has the potential to impact total power consumption. Similarly, the use of high-bandwidth optical wires or fast full-swing wires obviate the need for pipelined latches - a potential source for soft errors in processors. Thus, understanding wire technologies and exploiting them at the microarchitecture level can not only alleviate the wire delay problem but also open up new avenues to solve other major challenges faced by architects.

Simple wire design strategies can greatly influence a wire's properties. For example, by tuning wire width and spacing, wires can be designed with varying latency and bandwidth properties. Similarly, by tuning repeater size and spacing,

latency and energy of wires can be varied. To take advantage of VLSI techniques and better match the interconnect design to communication requirements, a heterogeneous interconnect is proposed, where every link consists of wires that are optimized for either latency, energy, or bandwidth.

1.3 Thesis Statement

The widening gap between transistor and wire delays necessitates careful analysis and better understanding of on-chip wires. VLSI techniques provide a number of different wires with varying delay, power, and area characteristics. The dissertation advocates exposing wire properties to architects and demonstrates that prudent management of wires at the microarchitectural level can improve both power and performance characteristics of future communication-bound processors.

In general, the effect of slow global wires is more pronounced in large structures in a processor. Since on-chip caches occupy more than 50% of the processor die [85] and up to 80% of their access time is attributed to wire delay [92], the dissertation focuses almost entirely on the prudent use of wires for cache accesses. When dealing with caches, there are three primary forms of communication entailed: wire delay to simply reach and access the Level-2 (L2) cache bank on a L2 cache hit, wire delay to ensure cache coherence when data are shared, and wire delay to access Level-1 (L1) cache data if a single-thread executes over a relatively large area (for example, in a high-ILP clustered design). This dissertation studies every aspect of communication happening within a cache hierarchy and innovates techniques to reduce their overheads. The first focus is on large lower level caches followed by coherence transactions and private L1 caches.

A large cache is typically partitioned into multiple banks and employs an on-chip network for address and data communication. Such a cache model is referred to as a Non-Uniform Cache Access (NUCA) architecture. Each bank is further partitioned into multiple subarrays with a h-tree network interconnecting these subarrays. The study demonstrates that a cache design that pays special attention to interbank network and h-tree design will perform significantly better in terms

of both performance and power. Novel design methodologies to identify optimal cache organizations are proposed and their evaluation shows that a design space exploration that includes both interconnect and cache parameters yields power- or performance-optimal cache organizations that are quite different from those assumed in prior studies.

The above study also gave key insights necessary to innovate new solutions. With interconnects having such a high influence, many architectural innovations are possible by leveraging interconnect properties. To further reduce the wire delay overhead and accelerate cache accesses, three novel cache pipelines are proposed that break the traditional sequential cache access pipeline. All the proposed techniques exploit the varying needs of address messages, data messages, and different bits in an address message and identify opportunities to carefully map them to a heterogeneous network to improve access time and reduce power consumption.

Having optimized regular cache reads and cache writes happening through lower level caches, as a next step, the focus is shifted to coherence transactions. With the proliferation of Chip-Multiprocessors (CMP), modern cache hierarchies incur an additional on-chip overhead of maintaining coherence among different private caches. These coherence messages involve on-chip communication with latencies that are projected to grow to tens of cycles in future billion transistor architectures [3]. A heterogeneous interconnect is very powerful in reducing the overhead of these transactions. For example, when employing a directory-based protocol, on a cache write miss, the requesting processor may have to wait for data from the home node (a two hop transaction) and for acknowledgments from other sharers of the block (a three hop transaction). Since the acknowledgments are on the critical path and have low bandwidth needs, they can be mapped to wires optimized for delay, while the data block transfer is not on the critical path and can be mapped to wires that are optimized for low power.

Finally, techniques to optimize L1 cache access pipeline are explored. Typically, accesses to L1 caches are wire limited in an aggressive high ILP processor such as a clustered architecture. A clustered architecture consists of many small and

fast computational units connected by a communication fabric. Since a cluster has limited resources and functionality, it enables fast clocks, low power, and low design effort compared to a monolithic design. However, the uneven scaling of transistors and wires makes communication between different clusters a major bottleneck. The dissertation shows that employing heterogeneous interconnect in such an environment not only helps alleviate communication delay but also improves intercluster bandwidth and interconnect power dissipation.

To expose interconnect properties to architects and strengthen future research methodologies, the proposed NUCA design methodologies along with interconnect models are incorporated into a publicly available, open-source cache modeling tool called CACTI. CACTI 6.0 [92], released as a part of this effort, focuses on design of large caches with special emphasis on interconnect design. Two major extentions to the tool were made: first, the ability to model different types of wires, such as RC-based wires with different power/delay characteristics and differential low-swing buses; second, the ability to model scalable Non-uniform Cache Access (NUCA). In addition to adopting the state-of-the-art design space exploration strategies for NUCA, the exploration is also enhanced by considering on-chip network contention and a wider spectrum of wiring and routing choices. Other key contributions to the tool include a dramatically improved search space along with facilities to perform trade-off analysis.

1.4 Dissertation Overview

The dissertation is organized as follows. Chapter 2 studies the design space for on-chip wires and discusses their analytical models. In Chapter 3, a novel methodology is discussed to model NUCA caches and describe CACTI 6.0 features in detail. Chapter 4 introduces new cache pipelines that leverage heterogeneous interconnects to accelerate cache accesses. Following this, Chapter 5 describes the application of heterogeneous interconnect in optimizing inter-core communications and reducing coherence overhead. Finally, Chapter 6 studies heterogeneous in-

terconnect configuration in a clustered architecture, followed by the conclusion in Chapter 7.

CHAPTER 2

DESIGN SPACE OF ON-CHIP WIRES

This chapter provides a quick overview of factors that influence wire properties and describes different wire implementations for a heterogeneous network. The second half of the chapter details analytical models employed for delay and power calculation of various full-swing and low-swing wires employed in a heterogeneous network.

2.1 Wire Implementations with Varying Characteristics

The delay of a wire is a function of the RC time constant (R is resistance and C is capacitance). The resistance per unit length of the wire can be expressed by the following equation [54]:

$$R_{wire} = \frac{\rho}{(thickness\ -\ barrier)(width\ -\ 2\ barrier)} \tag{2.1}$$

Thickness and *width* represent the geometrical dimensions of the wire cross-section, *barrier* represents the thin barrier layer around the wire to prevent copper from diffusing into surrounding oxide, and ρ is the material resistivity. The capacitance per unit length can be modeled by four parallel-plate capacitors for each side of the wire and a constant for fringing capacitance [54]:

$$C_{wire} = \epsilon_0(2K\epsilon_{horiz}\frac{thickness}{spacing} + 2\epsilon_{vert}\frac{width}{layerspacing})$$

$$+fringe(\epsilon_{horiz}, \epsilon_{vert}) \tag{2.2}$$

The potentially different relative dielectrics for the vertical and horizontal capacitors are represented by ϵ_{horiz} and ϵ_{vert}, K accounts for Miller-effect coupling capacitances, *spacing* represents the gap between adjacent wires on the same metal layer, and *layerspacing* represents the gap between adjacent metal layers. Now techniques that enable wires with varying properties are examined.

2.1.1 Wire Width and Spacing

As can be seen from equation (2.1), increasing the width of the wire can significantly decrease resistivity, while also resulting in a modest increase in capacitance per unit length (equation (2.2)). Similarly, increasing the spacing between adjacent wires results in a drop in C_{wire}. By allocating more metal area per wire and increasing the wire width and spacing, the overall effect is that the product of R_{wire} and C_{wire} decreases, resulting in lower wire delays. The primary difference between wires in the different types of metal layers in modern processors is the wire width and spacing (in addition to the thickness). Ho et al. [54] report that a 10mm unbuffered wire at 180nm technology has delays of 57 FO4s, 23 FO4s, and 6 FO4s on local, semiglobal, and global wires. Thus, wire width and spacing are powerful parameters that can vary the latency by at least a factor of 10. However, wide wires are more suited for low-bandwidth traffic such as for clock and power distribution. If global communication involves the transfer of 64-byte data between cache banks or cores, employing 512 wide wires can have enormous area overheads. For a given metal area, the wider the wire, the fewer the number of wires that can be accommodated (see Figure 2.1). Hence, optimizing a wire for low delay by designing wide wires has a negative impact on bandwidth.

2.1.2 Repeater Size and Spacing

The resistance and capacitance of a wire are both linear functions of the wire length. Hence, the delay of a wire, which depends on the product of wire resistance and capacitance, is a quadratic function of wire length. A simple technique to overcome this quadratic dependence is to break the wire into multiple smaller

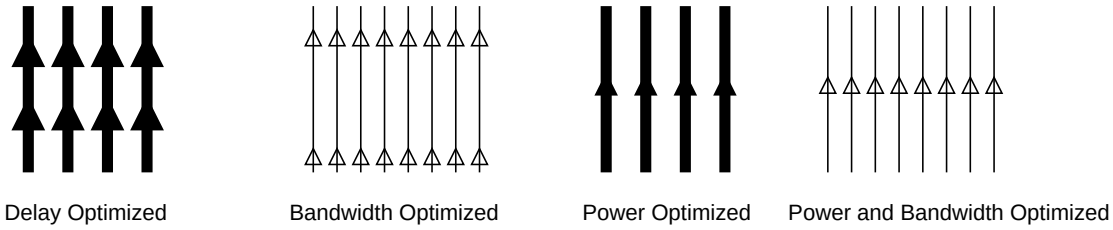


Figure 2.1. Examples of different wire implementations. Energy optimized wires have fewer and smaller repeaters, while bandwidth optimized wires have narrow widths and spacing.

segments and connect them with repeaters [9]. As a result, wire delay becomes a linear function of wire length and depends on the number of segments, the wire delay across each segment, and the logic delay across each repeater. Overall wire delay can be minimized by selecting optimal repeater sizes and spacing between repeaters [9] and this technique is commonly employed in modern-day processors. However, these repeaters have high overheads associated with them. Contacts have to be cut from the metal layer to the silicon substrate every time a logic element is introduced in the middle of a wire. The contacts and the transistors not only impose area overheads and routing constraints, but also impose high capacitive loads on the wires. Banerjee et al. [14, 15] report that sub-100nm designs will have over a million repeaters and that optimally sized repeaters are approximately 450 times the minimum sized inverter at that technology point.

Energy in the interconnect can be reduced by employing repeaters that are smaller than the optimally-sized repeaters and by increasing the spacing between successive repeaters (see Figure 2.1). This increases overall wire delay. Figure 2.2 shows the impact of repeater sizing and spacing on wire delay. Figure 2.3, shows the contours corresponding to the 2% delay penalty increments for different repeater configurations. Recently, Banerjee et al. [14] developed a methodology to estimate repeater size and spacing that minimizes power consumption for a fixed wire delay. They show that at 50nm technology, it is possible to design a repeater configuration such that the wire has twice the delay and $1/5^{th}$ the energy of a wire

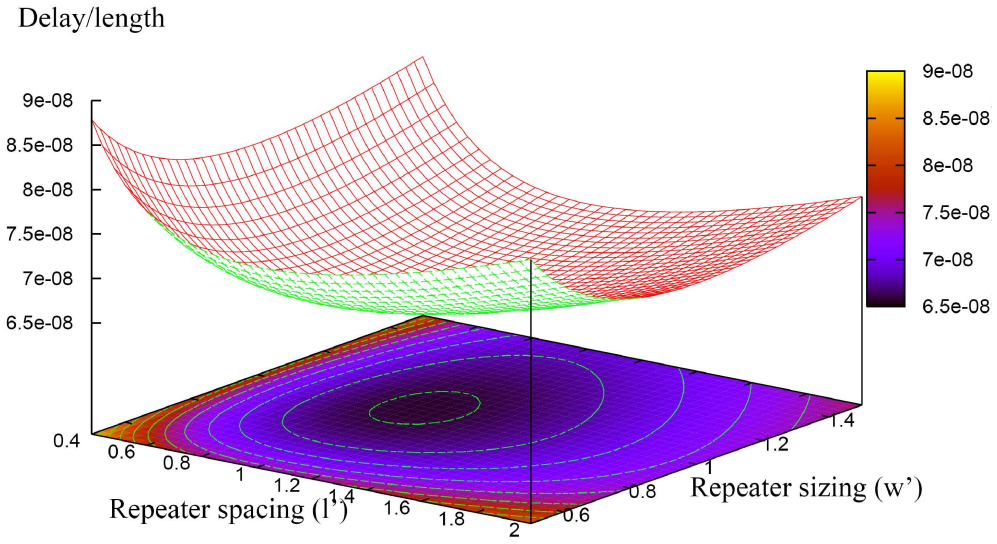


Figure 2.2. Effect of repeater spacing/sizing on wire delay.

that is delay-optimal. Thus, repeater size and spacing are parameters that can dramatically influence interconnect power and performance.

2.1.3 Low-swing Wires

One of the primary reasons for the high power dissipation of global wires is the full swing (0 V to V_{dd}) requirement imposed by the repeaters. While the repeater spacing and sizing adjustments reduce power overhead to some extent, the requirement is still relatively high. Low voltage swing alternatives represent another mechanism to vary the wire power, delay, and area trade-off. Reducing the voltage swing on global wires can result in a linear reduction in power. In addition, assuming a separate voltage source for low-swing drivers will result in a quadratic savings in power. However, these lucrative power savings are accompanied by many caveats. Since we can no longer use repeaters or latches, the delay of a low-swing wire increases quadratically with length. Since such a wire cannot be pipelined, they also suffer from lower throughput. A low-swing wire requires special transmitter and receiver circuits for signal generation and amplification. This not only increases

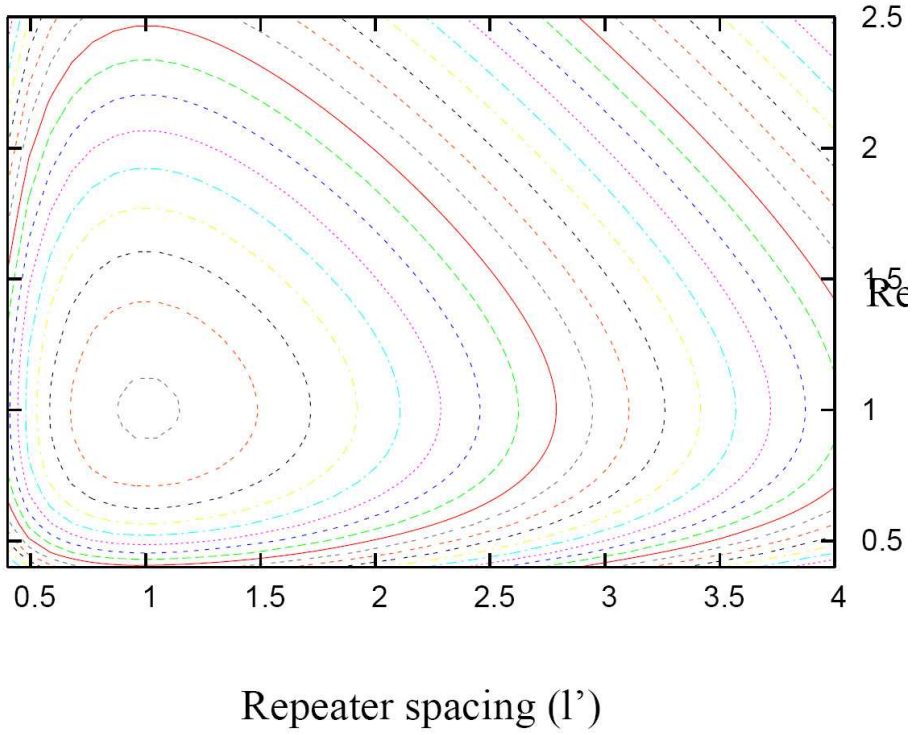


Figure 2.3. Contours for 2% delay penalty

the area requirement per bit, but also assigns a fixed cost in terms of both delay and power for each bit traversal. In spite of these issues, the power savings possible through low-swing signalling makes it an attractive design choice.

Due to fixed overhead associated with low-swing wires, their application is beneficial only for long on-chip wires. Hence, the study limits the application of low-swing wires to address and data network in large caches.

2.1.4 Transmission Lines

In future technologies, other promising wire implementations may become feasible, such as transmission lines [31, 41]. In a transmission line, the wire delay is determined by the time taken to detect a voltage ripple on the wire. This delay is determined by the LC time constant and the velocity of the ripple, which is a function of the speed of light in the dielectric surrounding the interconnect. A transmission line, therefore, enables very low wire latencies. For a wire to operate

as a transmission line, it must have very high width, thickness, horizontal and vertical spacing, and signal frequency. There are other implementation issues as well, such as the design of signal modulation and sensing circuits, reference planes above and below the metal layer, and shielding power and ground lines adjacent to each transmission line [20].

Because of the large area requirements and other associated costs, transmission lines have been sparsely used in modern processors, usually as single wires for clock distribution [86, 124, 128]. They have also been shown to work in other test CMOS chips [31, 43]. As we move to higher clock frequencies and increasing metal layers, transmission line implementations may become more practical and cost-effective. However, owing to the high area requirements per wire, transmission lines are likely to be feasible only for very low bandwidth communication. Thus, a transmission line represents another interesting wire implementation that trades off bandwidth for extremely low latencies. However, due to the complexity associated with transmission lines, they are not considered further in this dissertation.

2.2 Heterogeneous Interconnects

From the above discussion, it is clear that a large number of different wire implementations are possible, either by varying properties such as wire width/spacing and repeater size/spacing, or by employing transmission lines. In addition to traditional global wires that are sized and spaced to optimize delay (they are referred to here as *B-Wires*), there are at least three other interesting wire implementations that the architecture can benefit from:

- *P-Wires:* Wires that are power-optimal. The wires have longer delays as they employ small repeater sizes and wide repeater spacing.

- *W-Wires:* Wires that are bandwidth-optimal. The wires have minimum width and spacing and have longer delays.

- *L-Wires:* Wires that are latency-optimal. These wires employ very wide width and spacing and have low bandwidth (potentially, a network with fewer than 20 bits).

To limit the range of possibilities, *P-Wires* and *W-Wires* can be combined to form a single wire implementation with minimum width and spacing and with small repeater sizes and wide repeater spacing. Such wires have poor delay characteristics, but allow low power and high bandwidth (referred to as *PW-Wires*). Modern technology allows more than 10 layers of metal in a processor [69]. The pitch of wires in each layer is determined by technology constraints (such as minimum via size that can tolerate variation, minimum metal width, etc.) and metal area budget. Typically low level metal layers constitute wires with smaller pitch and are employed for macro-level wiring. The latencies of these wires are a fraction of a cycle and their lengths scale down with process technology. The growing disparity between transistor and wire delay is more pronounced in top level wires that are used for time-critical communications across the chip. This study assumes that B and L wires are implemented in 8X (global) plane and PW wires are placed in 4X (semiglobal) plane. However, the proposed implementations are possible in any metal plane.

2.3 Analytical Models

The following sections discuss analytical delay and power models for different wires. All the process specific parameters required for calculating the transistor and wire parasitics are obtained from ITRS [7].

2.3.1 Global Wires

For a long repeated wire, the single pole time constant model for the interconnect fragment shown in Figure 2.4 is given by,

$$\tau = (\frac{1}{l} r_s(c_0 + c_p) + \frac{r_s}{s} C_{wire} + R_{wire} s c_0 + 0.5 R_{wire} C_{wire} l) \tag{2.3}$$

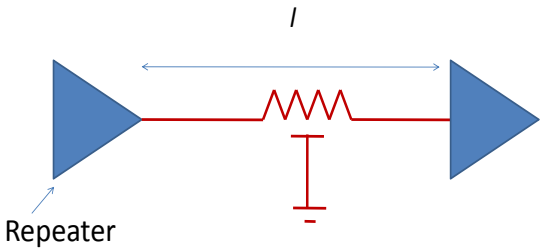


Figure 2.4. Interconnect segment

In the above equation, c_0 is the capacitance of the minimum sized repeater, c_p is its output parasitic capacitance, r_s is its output resistance, l is the length of the interconnect segment between repeaters, and s is the size of the repeater normalized to the minimum value. The values of c_0, c_p, and r_s are constant for a given process technology. Wire parasitics R_{wire} and C_{wire} represent resistance and capacitance per unit length. The optimal repeater sizing and spacing values can be calculated by differentiating equation (2.3) with respect to s and l and equating it to zero.

$$L_{optimal} = \sqrt{\frac{2r_s(c_0 + c_p)}{R_{wire}C_{wire}}} \tag{2.4}$$

$$S_{optimal} = \sqrt{\frac{r_s C_{wire}}{R_{wire}c_0}} \tag{2.5}$$

The delay value calculated using the above $L_{optimal}$ and $S_{optimal}$ is guaranteed to have minimum value.

The total power dissipated is the sum of three main components (equation (2.6)) [14].

$$P_{total} = P_{switching} + P_{short-circuit} + P_{leakage} \tag{2.6}$$

The dynamic and leakage components of the repeaters in a wire are computed using equations (2.8) and (2.10).

$$P_{dynamic} = \alpha V_{DD}^2 f_{clock}(\frac{S_{optimal}}{L_{optimal}}(c_p + c_0) + c) \quad (2.7)$$

$$+(\alpha V_{DD} W_{min} I_{SC} f_{clock} log_e 3) S_{optimal} \frac{\tau}{L_{optimal}} \quad (2.8)$$

f_{clock} is the operating frequency, W_{min} is the minimum width of the transistor, I_{SC} is the short-circuit current, and the value $(\tau/L)_{optimal}$ can be calculated from equation (2.9).

$$(\frac{\tau}{L})_{optimal} = 2\sqrt{r_s c_0 r c}\left(1 + \sqrt{0.5 * \left(1 + \frac{c_p}{c_0}\right)}\right) \quad (2.9)$$

$$P_{leakage} = \frac{3}{2} V_{DD} I_{leak} W_n S_{optimal} \quad (2.10)$$

I_{leak} is the leakage current and W_n is the minimum width of the nMOS transistor.

With the above equations, it is possible to compute the delay and power for global and semiglobal wires. Wires faster than global wires can be obtained by increasing the wire width and spacing between the wires. Wires whose repeater spacing and sizing are different from equation (2.4) and (2.5) will incur a delay penalty. For a given delay penalty, the power optimal repeater size and spacing can be obtained from the contour shown in Figure 2.3. The actual calculation involves solving a set of differential equations [14].

2.3.2 Low-swing Wires

A low-swing interconnect system consists of three main components: (1) a transmitter that generates and drives the low-swing signal, (2) twisted differential wires, and (3) a receiver amplifier.

- **Transmitter**

  For an RC tree with a time constant τ, the delay of the circuit for an input with finite rise time is given by equation (2.11)

$$delay_r = t_f \sqrt{[\log \frac{v_{th}}{V_{dd}}]^2 + 2t_{rise} b(1 - \frac{v_{th}}{Vdd})/t_f} \tag{2.11}$$

  where t_f is the time constant of the tree, v_{th} is the threshold voltage of the transistor, t_{rise} is the rise time of the input signal, and b is the fraction of the input swing in which the output changes (b is assumed to be 0.5).

  For falling input, the equation changes to

$$delay_f = t_f \sqrt{[\log(1 - \frac{v_{th}}{V_{dd}})]^2 + \frac{2t_{fall} b v_{th}}{t_f V_{dd}}} \tag{2.12}$$

  where t_{fall} is the fall time of the input. For the falling input, a value of 0.4 is assumed for b [125].

  To get a reasonable estimate of the initial input signal rise/fall time, two inverters connected in series are considered. Let d be the delay of the second inverter. The t_{fall} and t_{rise} values for the initial input can be approximated to

$$t_{fall} = \frac{d}{1 - v_{th}}$$

$$t_{rise} = \frac{d}{v_{th}}$$

  For the transmitter circuit shown in Figure 2.5, the model proposed by Ho et al. [55] is employed.

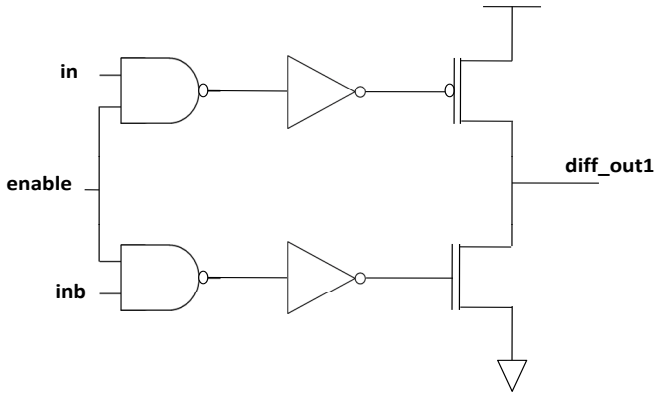


Figure 2.5. Low-swing transmitter (actual transmitter has two such circuits to feed the differential wires)

The total delay of the transmitter is given by,

$$t_{delay} = nand_{delay} + inverter_{delay} + driver_{delay} \tag{2.13}$$

Each gate in the above equation (*nand*, *inverter*, and *driver*) can be reduced to a simple RC tree. Later, a Horowitz approximation [125] is applied to calculate the delay of each gate. The power consumed in different gates can be derived from the input and output parasitics of the transistors.

NAND gate: The equivalent resistance and capacitance values of a NAND gate is given by,

$$R_{eq} = 2 * R_{nmos}$$

$$C_{eq} = 2 * C_{Pdrain} + 1.5 * C_{Ndrain} + C_L$$

where C_L is the load capacitance of the NAND gate and is equal to the input capacitance of the next gate. The value of C_L is equal to $INV_{size} * (C_{Pgate} +$

C_{Ngate}) where equal to the input capacitance of the next gate. The value of C_L is equal to $INV_{size} * (C_{Pgate} + C_{Ngate})$ where INV_{size} is the size of the inverter whose calculation is discussed later in this section.

NOTE: The drain capacitance of a transistor is highly nonlinear. In the above equation for C_{eq}, the effective drain capacitance of two nMOS transistors connected in series is approximated to 1.5 times the drain capacitance of a single nMOS transistor.

$$\tau_{nand} = R_{eq} * C_{eq}$$

Using the τ_{nand} and t_{rise} values in equation (2.12), $nand_{delay}$ can be calculated. Power consumed by the NAND gate is given by

$$P_{nand} = C_{eq} * V_{dd}^2$$

The fall time (t_{fall}) of the input signal to the next stage (NOT gate) is given by

$$t_{fall} = nand_{delay}(\frac{1}{1 - v_{th}})$$

Driver: To increase the energy savings in low-swing model, a separate low voltage source for driving low-swing differential wires is assumed. The size of these drivers depends on its load capacitance, which in turn depends on the length of the wire. To calculate the size of the driver, the drive resistance of the nMOS transistors for a fixed desired rise time of eight F04 is first calculated.

$$R_{drive} = \frac{-Risetime}{C_L * ln(0.5)}$$

$$W_{dr} = \frac{R_m}{R_{drive}} * W_{min}$$

In the above equation, C_L is the sum of capacitance of the wire and input capacitance of the sense amplifier. R_m is the drive resistance of a minimum sized nMOS transistor and W_{min} is the width of the minimum sized transistor.

From the R_{drive} value, the actual width of the pMOS transistor can be calculated. The model limits the transistor width to 100 times the minimum size.

NOTE: The driver resistance R_{drive} calculated above is valid only if the supply voltage is set to full V_{dd}. Since low-swing drivers employ a separate low voltage source, the actual drive resistance of these transistors will be greater than the pMOS transistor of the same size driven by the full V_{dd}. Hence, the R_{drive} value is multiplied with an adjustment factor RES_ADJ to account for the poor driving capability of the pMOS transistor. Based on the SPICE simulation, RES_ADJ value is calculated to be 8.6.

NOT gate: The size of the NOT gate is calculated by applying the method of logical effort. Consider the NAND gate connected to the NOT gate that drives a load of C_L, where C_L is equal to the input capacitance of the driver.

$$path\_effort = \frac{C_L}{C_{Ngate} + C_{Pgate}}$$

The delay will be minimum when the effort in each stage is the same.

$$stage\_effort = \sqrt{(4/3) * path\_effort}$$

$$C_{NOT\_in} = \frac{(4/3) * C_L}{stage\_effort}$$

$$INV_{size} = \frac{C_{NOT_in}}{C_{Ngate} + C_{Pgate}}$$

Using the above inverter size, the equivalent resistance and the capacitance of the gate can be calculated.

$$R_{eq} = R_{pmos}$$

$$C_{eq} = C_{Pdrain} + C_{Ndrain} + C_L$$

where C_L for the inverter is equal to (C_{Ngate}).

$$\tau_{not} = R_{eq} * C_{eq}$$

Using the above τ_{not} and t_{fall} values, not_{delay} can be calculated. Energy consumed by this NOT gate is given by

$$E_{not} = C_{eq} * V_{dd}^2$$

The rise time for the next stage is given by

$$t_{rise} = \frac{not_{delay}}{v_{th}}$$

- **Differential Wires**

  To alleviate the high delay overhead of the un-repeated low-swing wires, similar to differential bitlines, pre-emphasis and pre-equalization optimizations are employed. In pre-emphasis, the drive voltage of the driver is maintained at higher voltage than low-swing voltage. By overdriving these wires, it takes only a fraction of time constant to develop the differential voltage. In pre-equalization, after a bit traversal, the differential wires are shorted to recycle the charge. Developing a differential voltage on pre-equalized wires takes less time compared to the wires with opposite polarity.

  The following equations present the time constant and capacitance values of the segments that consist of low-swing drivers and wires.

$$t_{driver} = (R_{driver} * (C_{wire} + 2 * C_{drain}) + R_{wire} C_{wire}/2 + (R_{driver} + R_{wire}) * C_{sense\_amp}) \tag{2.14}$$

The C_{wire} and R_{wire} in the above equation represents resistance and capacitance parasitics of the low-swing wire. R_{driver} and C_{drain} are resistance and drain capacitance of the driver transistors. The pre-equalization and pre-emphasis optimizations bring down this time constant to 35% of the above value.

The total capacitance of the low-swing segment is given by

$$C_{load} = C_{wire} + 2 * C_{drain} + C_{sense\_amp}$$

The dynamic energy due to charging and discharging of differential wires is given by

$$C_{load} * V_{overDrive} * V_{lowswing}$$

For our evaluations, an overdrive voltage of 400mV and a low swing voltage of 100mV is assumed.

- **Sense Amplifier**

  Figure 2.6 shows the cross-coupled inverter sense amplifier circuit used at the receiver. The delay and power values of the sense amplifier were directly calculated from SPICE simulation and tabulated in Table 2.1.

2.4 Summary

This chapter examined different wire implementations possible in a network. It showed that wires come in a variety of flavors with different power, delay, and bandwidth characteristics. Then, heterogeneous interconnect fabric is defined that consists of three different types of wires: low-latency, low-bandwidth wires (L-wires), traditional base case wires (B-wires), and power and bandwidth efficient wires (PW-wires). The subsequent chapters will study how interconnect choices influence the design of large caches and explore techniques that leverage different types of wires to improve power-delay characteristics of on-chip caches.

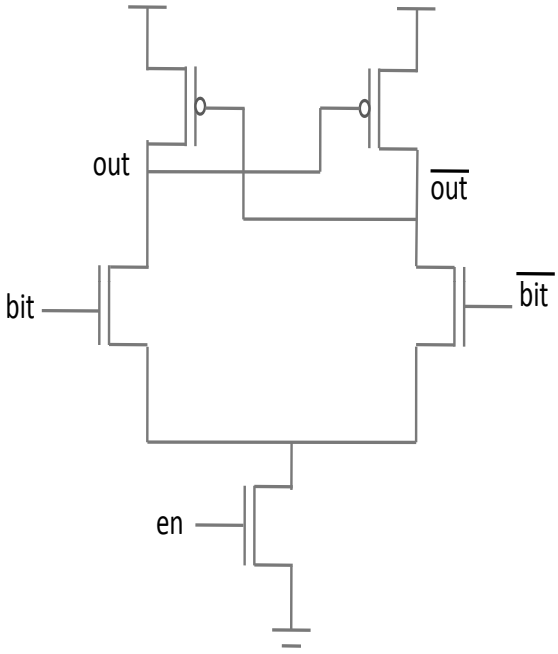


Figure 2.6. Sense-amplifier circuit

Table 2.1. Sense-amplifier delay and energy values for different process technologies

Technology	Delay (ps)	Energy (fJ)
90nm	279	14.7
68nm	200	5.7
45nm	38	2.7
32nm	30	2.16

CHAPTER 3

INTERCONNECT DESIGN FOR LARGE CACHES

Chapter 2 discussed various types of wires along with their power, delay, and bandwidth characteristics. This chapter demonstrates how the choice of these wires influences the design of on-chip caches. To design scalable large caches, many recent proposals advocate splitting the cache into a large number of banks and employing a network-on-chip (NoC) to allow fast access to nearby banks (referred to as Non-Uniform Cache Access (NUCA) Architecture).

Most studies on NUCA organizations have assumed a generic NoC and focused on logical policies for cache block placement, movement, and search. Since wire/router delay and power are major limiting factors in modern processors, this work focuses on interconnect design and its influence on NUCA performance and power. With interconnect overheads appropriately accounted for, the optimal cache organization is typically very different from that assumed in prior NUCA studies. In an effort to strengthen future NUCA design methodologies and to make interconnect models explicit to architects, the proposed techniques are incorporated into an open-source cache modeling tool called CACTI.

CACTI 6.0, released as part of this work, is provided with two major extensions: first, the ability to model different types of wires, such as RC-based wires with different power/delay characteristics and differential low-swing buses are added to the tool; second, the tool is enhanced to model Non-uniform Cache Access (NUCA). In addition to adopting state-of-the-art design space exploration strategies for NUCA, the exploration is also enhanced by considering on-chip network contention and a wider spectrum of wiring and routing choices. At the end of the chapter, a

validation analysis of the new tool is presented along with a case study to showcase how the tool can improve architecture research methodologies.

3.1 Need for an Enhanced Cache Modeling Tool

Multi-core processors will incorporate large and complex cache hierarchies. The Intel Montecito employs two 12 MB private L3 caches, one for each core [85], and there is speculation that future 3D processors may dedicate an entire die for large SRAM caches or DRAM main memory [23, 78, 103]. Therefore, it is expected that future processors will have to intelligently manage many mega-bytes of on-chip cache. Future research will likely explore architectural mechanisms to (i) organize the L2 or L3 cache into shared/private domains, (ii) move data to improve locality and sharing, (iii) optimize the network parameters (topology, routing) for efficient communication between cores and cache banks. Examples of on-going research in these veins include [19, 21, 30, 33, 34, 45, 57, 59, 65, 75, 109, 131].

Many cache evaluations employ the CACTI cache access modeling tool [126] to estimate delay, power, and area for a given cache size. The CACTI estimates are invaluable in setting up baseline simulator parameters, computing temperatures of blocks neighboring the cache, evaluating the merits/overheads of novel cache organizations (banking, reconfiguration, additional bits of storage), etc. While CACTI 5.0 produces reliable delay/power/area estimates for moderately sized caches, it does not model the requirements of large caches in sufficient detail. Besides, the search space of the tool is limited and, hence, so is its application in power/performance trade-off studies. With much of future cache research focused on multimegabyte cache hierarchy design, this is a serious short-coming. Hence, in this work, the CACTI tool is extended in many ways, with the primary goal of improving the fidelity of its large cache estimates. The tool can also aid in trade-off analysis; for example, with a comprehensive design space exploration, CACTI 6.0 can identify cache configurations that consume three times less power for about a 25% delay penalty.

The main enhancement provided in CACTI 6.0 is a very detailed modeling of the interconnect between cache banks. A large cache is typically partitioned into many smaller banks and an inter-bank network is responsible for communicating addresses and data between banks and the cache controller. Earlier versions of CACTI have employed a simple H-tree network with global wires and have assumed uniform access times for every bank (a uniform cache access architecture, referred to as UCA). Recently, nonuniform cache architectures (NUCA [65]) have also been proposed that employ a packet-switched network between banks and yield access times that are a function of where data blocks are found (not a function of the latency to the most distant bank). Support for such an architecture within CACTI is added.

As the size of the cache scales up, irrespective of using a packet-switched or H-tree network, the delay and power of the network components dominate the overall cache access delay and power. Figure 3.1 shows that the H-tree of the CACTI 5.0 model contributes an increasing percentage to the overall cache delay as the cache size is increased from 2 to 32 MB. Its contribution to total cache power is also sizable: around 50%. As the cache size increases, the bitline power component also grows. Hence, the contribution of H-tree power as a percentage remains roughly constant. The inter-bank network itself is sensitive to many parameters, especially the wire signaling strategy, wire parameters, topology, router configuration, etc. The new version of the tool carries out a design space exploration over these parameters to estimate a cache organization that optimizes a combination of power/delay/area metrics for UCA and NUCA architectures. Network contention plays a nontrivial role in determining the performance of an on-chip network design. To consider its effect, the design space exploration is augmented with empirical data on network contention.

Components of the tool are partially validated against detailed Spice simulations. Finally, an example case study is presented to demonstrate how the tool can facilitate architectural evaluations.

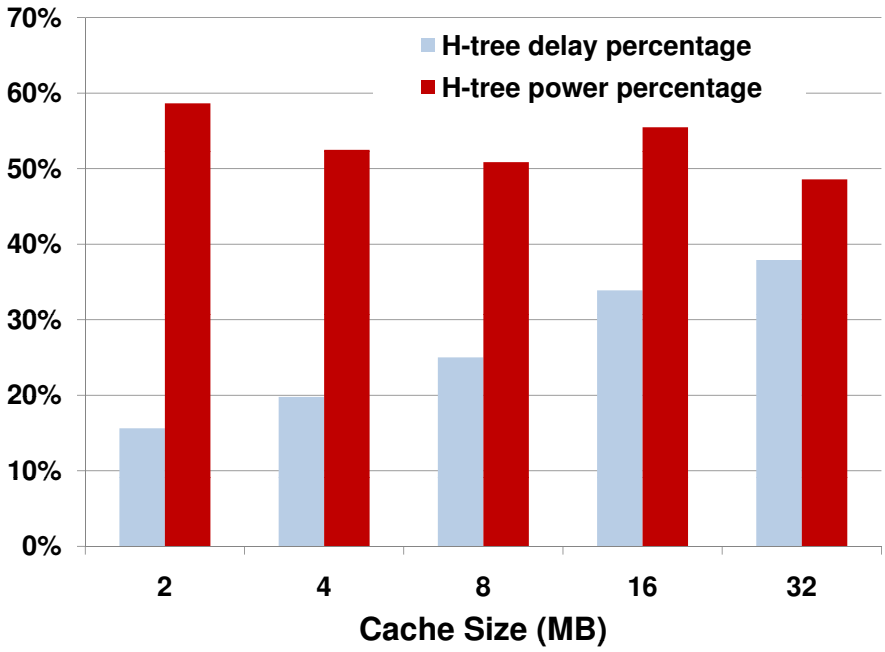


Figure 3.1. Contribution of H-tree network to overall cache delay and power

3.2 CACTI 6.0

Figure 3.2 shows the basic logical structure of a uniform cache access (UCA) organization. The address is provided as input to the decoder, which then activates a wordline in the data array and tag array. The contents of an entire row (referred to as a *set*) are placed on the bitlines, which are then sensed. The multiple tags thus read out of the tag array are compared against the input address to detect if one of the ways of the set does contain the requested data. This comparator logic drives the multiplexor that finally forwards at most one of the ways read out of the data array back to the requesting processor.

The CACTI cache access model [112] takes in the following major parameters as input: cache capacity, cache block size (also known as cache line size), cache associativity, technology generation, number of ports, and number of independent banks (not sharing address and data lines). As output, it produces the cache configuration that minimizes delay (with a few exceptions), along with its power

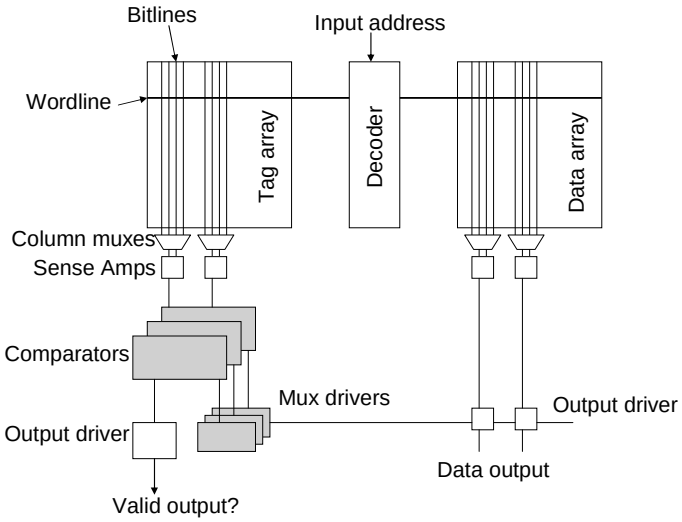


Figure 3.2. Logical organization of the cache (from CACTI 3.0 [108])

and area characteristics. CACTI models the delay, power, and area of eight major cache components: decoder, wordline, bitline, senseamp, comparator, multiplexer, output driver, and inter-bank wires. The wordline and bitline delays are two of the most significant components of the access time. The wordline and bitline delays are quadratic functions of the width and height of each array, respectively. In practice, the tag and data arrays are large enough that it is inefficient to implement them as single large structures. Hence, CACTI partitions each storage array (in the horizontal and vertical dimensions) to produce smaller *subarrays* and reduce wordline and bitline delays. The bitline is partitioned into *Ndbl* different segments, the wordline is partitioned into *Ndwl* segments, and so on. Each subarray has its own decoder and some central predecoding is now required to route the request to the correct subarray. The most recent version of CACTI (version 5.0) employs a model for semiglobal (intermediate) wires and an H-tree network to compute the delay between the predecode circuit and the furthest cache subarray. CACTI carries out an exhaustive search across different subarray counts (different values of Ndbl, Ndwl, etc.) and subarray aspect ratios to compute the cache organization with optimal total delay. Typically, the cache is organized into a handful of subarrays. An example of the cache's physical structure is shown in Figure 3.3.

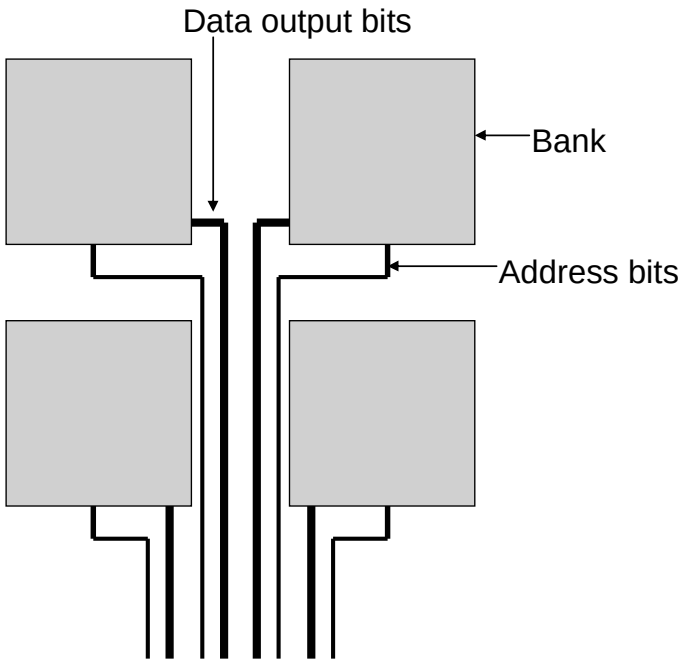


Figure 3.3. Physical organization of the cache (from CACTI 3.0 [108])

3.2.1 Interconnect Modeling for UCA Caches

As already shown in Figure 3.1, as cache size increases, the interconnect (within the H-tree network) plays an increasingly greater role in terms of access time and power. The interconnect overhead is impacted by (i) the number of subarrays, (ii) the signaling strategy, and (iii) the wire parameters. While prior versions of CACTI iterate over the number of subarrays by exploring different values of Ndbl, Ndwl, Nspd, etc., the model is restricted to a single signaling strategy and wire type (global wire). Thus, the design space exploration sees only a modest amount of variation in the component that dominates the overall cache delay and power. Therefore, the exploration is extended to also include a low-swing differential signaling strategy as well as the use of local and *fat* wires whose characteristics are shown in Figure 3.4 and 3.5.

By examining these choices and carrying out a more comprehensive design space exploration, CACTI 6.0 is able to identify cache organizations that better meet the user-specified metrics. Figure 3.6 shows a power-delay curve where each point

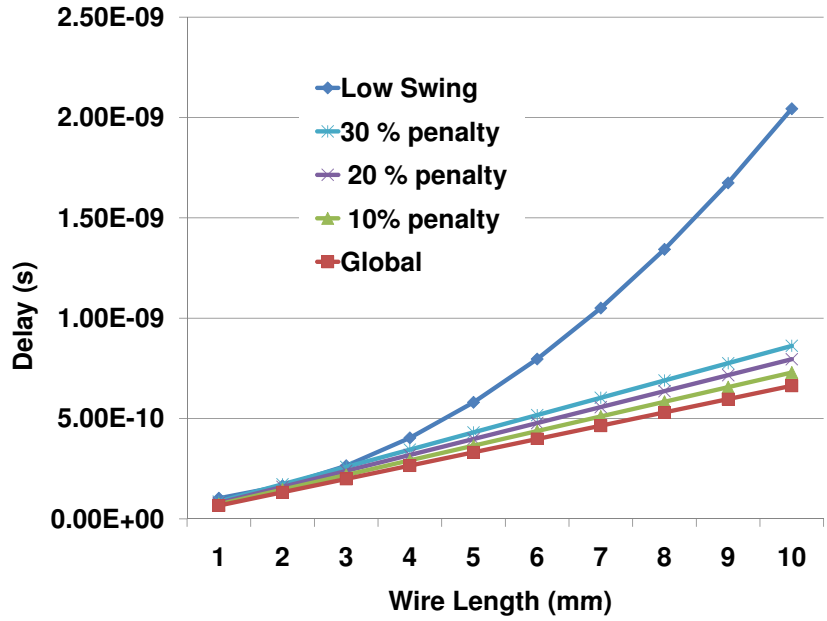


Figure 3.4. Delay characteristics of different wires

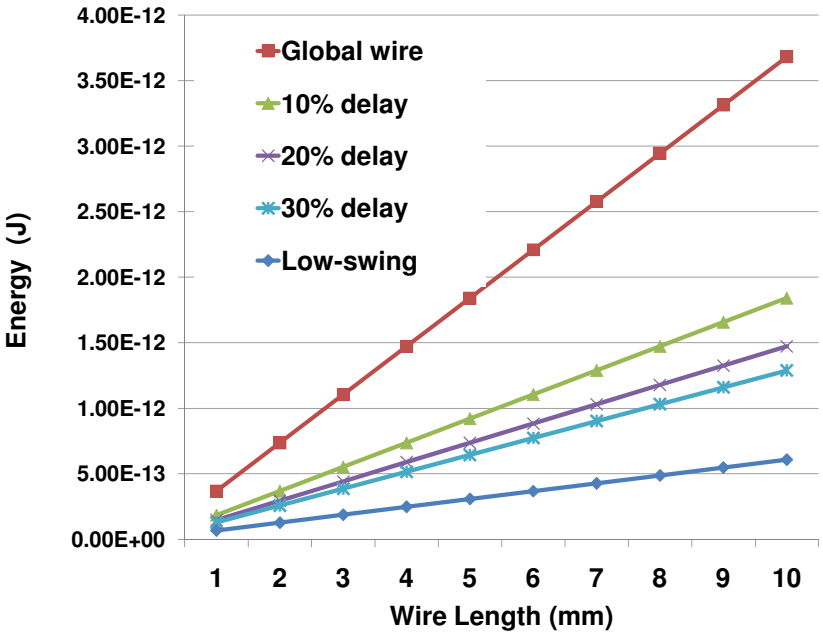


Figure 3.5. Energy characteristics of different wires

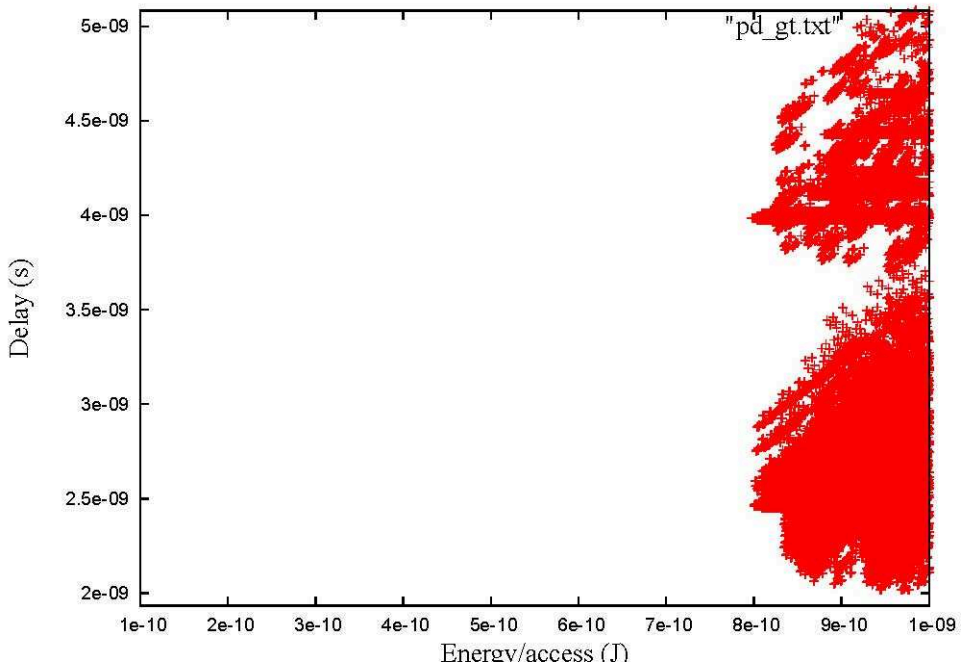


Figure 3.6. Design space exploration with global wires

represents one of the many hundreds of cache organizations considered by CACTI 5.0. The red points (bottom region) represent the cache organizations that would have been considered by CACTI 5.0 with its limited design space exploration with a single wire type (global wire with delay-optimal repeaters). The yellow points (middle region) in Figure 3.7 represent cache organizations with different wire types that are considered by CACTI 6.0. Clearly, by considering the trade-offs made possible with wires and expanding the search space, CACTI 6.0 is able to identify cache organizations with very relevant delay and power values.

The choice of an H-tree network for CACTI 5.0 (and earlier versions of CACTI) was made for the following reason: it enables uniform access times for each bank, which in turn, simplifies the pipelining of requests across the network. Since low-swing wires cannot be pipelined and since they better amortize the transmitter/receiver overhead over long transfers, a different network style is adopted when using low-swing wires. Instead of the H-tree network, a collection of simple broadcast buses is adopted that span across all the banks (each bus is shared by

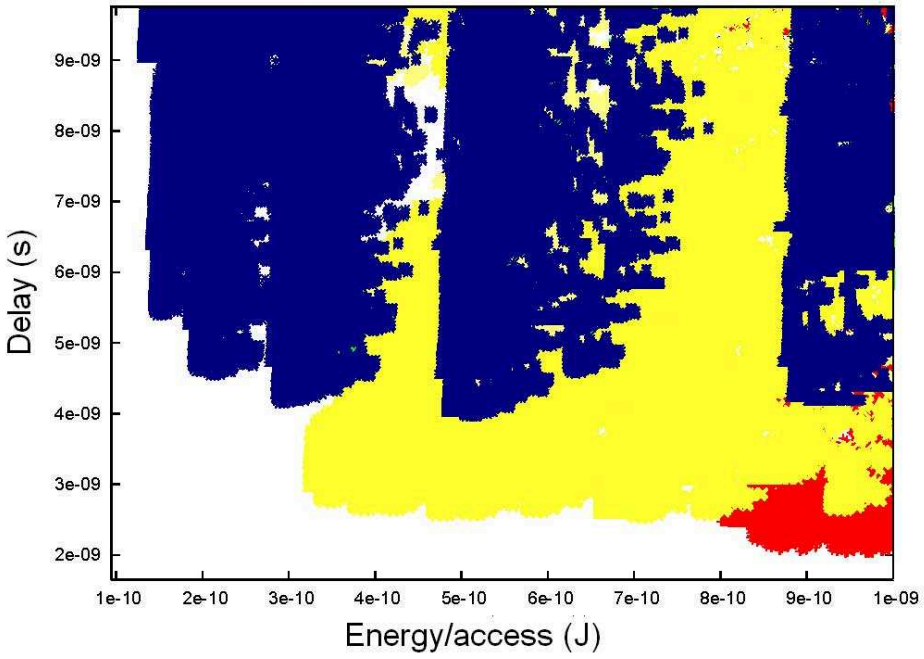


Figure 3.7. Design space exploration with full-swing global wires (red, bottom region), wires with 30% delay penalty (yellow, middle region), and differential low-swing wires (blue, top region)

half the banks in a column – an example with eight banks is shown in Figure 3.8). The banks continue to have uniform access times, as determined by the worst-case delay. Since the bus is not pipelined, the wire delay limits the throughput as well and decreases the operating frequency of the cache. The cycle time of a cache is equal to the maximum delay of a segment that cannot be pipelined. Typically, the sum of bitline and sense amplifier delay decides the cycle time of a cache. In a low-swing model, the cycle time is determined by the maximum delay of the low-swing bus. Low-swing wires with varying width and spacing are also considered that further play into the delay/power/area trade-offs.

With low-swing wires included in the CACTI design space exploration, the tool is able to identify many more points that yield low power at a performance and area cost. The blue points (top region) in Figure 3.7 represent the cache organizations considered with low-swing wires. Thus, by leveraging different wire properties, it

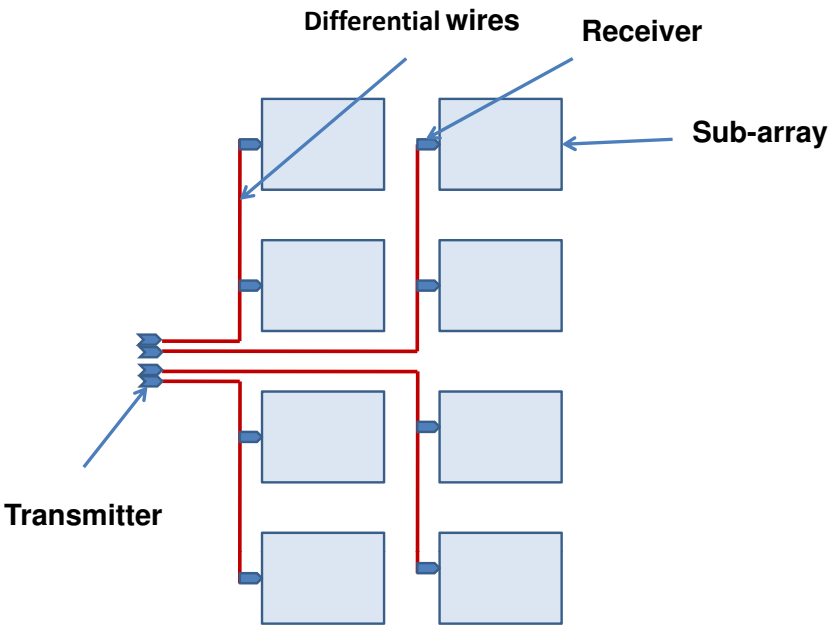


Figure 3.8. 8-bank data array with a differential low-swing broadcast bus

is possible to generate a broad range of cache models with different power/delay characteristics.

3.2.2 NUCA Modeling

The UCA cache model discussed so far has an access time that is limited by the delay of the slowest sub-bank. A more scalable approach for future large caches is to replace the H-tree bus with a packet-switched on-chip grid network. The latency for a bank is determined by the delay to route the request and response between the bank that contains the data and the cache controller. Such a NUCA model was first proposed by Kim et al. [65] and has been the subject of many architectural evaluations. Therefore, CACTI is extended to support such NUCA organizations as well.

The tool first iterates over a number of bank organizations: the cache is partitioned into 2^N banks (where N varies from 1 to 12); for each N, the banks

are organized in a grid with 2^M rows (where M varies from 0 to N). For each bank organization, CACTI 5.0 is employed to determine the optimal subarray partitioning for the cache within each bank. Each bank is associated with a router. The average delay for a cache access is computed by estimating the number of network hops to each bank, the wire delay encountered on each hop, and the cache access delay within each bank. It is further assumed that each traversal through a router takes up R cycles, where R is a user-specified input. Router pipelines can be designed in many ways: a four-stage pipeline is commonly advocated [42], and recently, speculative pipelines that take up three, two, and one pipeline stage have also been proposed [42, 90, 98]. While the user is given the option to pick an aggressive or conservative router, the tool defaults to employing a moderately aggressive router pipeline with three stages.

More partitions lead to smaller delays (and power) within each bank, but greater delays (and power) on the network (because of the constant overheads associated with each router and decoder). Hence, the above design space exploration is required to estimate the cache partition that yields optimal delay or power. The above algorithm was recently proposed by Muralimanohar and Balasubramonian [91]. The algorithm is further extended in the following ways.

First, different wire types for the links between adjacent routers are explored. These wires are modeled as low-swing differential wires as well as local/global/fat wires to yield many points in the power/delay/area spectrum.

Second, different types of routers are modeled. The sizes of buffers and virtual channels within a router have a major influence on router power consumption as well as router contention under heavy load. By varying the number of virtual channels per physical channel and the number of buffers per virtual channel, different points on the router power-delay trade-off curve could be achieved.

Third, contention in the network in much greater detail is modeled. This itself has two major components. If the cache is partitioned into many banks, there are more routers/links on the network and the probability of two packets conflicting at a router decrease. Thus, a many-banked cache is more capable of meeting the

bandwidth demands of a many-core system. Further, certain aspects of the cache access within a bank cannot be easily pipelined. The longest such delay within the cache access (typically the bitline and sense-amp delays) represents the cycle time of the bank – it is the minimum delay between successive accesses to that bank. A many-banked cache has relatively small banks and a relatively low cycle time, allowing it to support a higher throughput and lower wait-times once a request is delivered to the bank. Both of these two components (lower contention at routers and lower contention at banks) tend to favor a many-banked system. This aspect is also included in estimating the average access time for a given cache configuration.

The contention values for each considered NUCA cache organization are empirically estimated for typical workloads and incorporated into CACTI 6.0 as look-up tables. For the grid topologies considered (for different values of N and M), L2 requests originating from single-core, two-core, four-core, eight-core, and sixteen-core processors are simulated. Each core executes a mix of programs from the SPEC benchmark suite. The benchmark set is divided into four categories, as described in Table 3.1.

For every CMP organization, four sets of simulations are run, corresponding to each benchmark set tabulated. The generated cache traffic is then modeled on a detailed network simulator with support for virtual channel flow control. Details of the architectural and network simulator are listed in Table 3.2. The contention value (averaged across the various workloads) at routers and banks is estimated for each network topology and bank cycle time. Based on the user-specified inputs, the

Table 3.1. Benchmark sets

Memory intensive benchmarks	applu, fma3d, swim, lucas equake, gap, vpr, art
L2/L3 latency sensitive benchmarks	ammp, apsi, art, bzip2, crafty, eon, equake, gcc
Half latency sensitive & half non-latency sensitive benchmarks	ammp, applu, lucas, bzip2 crafy, mgrid, mesa, gcc
Random benchmark set	Entire SPEC suite

Table 3.2. Simplescalar simulator parameters

Fetch queue size	64
Branch predictor	comb. of bimodal and 2-level
Bimodal predictor size	16K
Level 1 predictor	16K entries, history 12
Level 2 predictor	16K entries
BTB size	16K sets, 2-way
Branch mispredict penalty	at least 12 cycles
Fetch width	8 (across up to 2 basic blocks)
Dispatch and commit width	8
Issue queue size	60 (int and fp, each)
Register file size	100 (int and fp, each)
Re-order Buffer size	80
L1 I-cache	32KB 2-way
L1 D-cache	32KB 2-way set-associative,
L2 cache	32MB 8-way SNUCA 3 cycles, 4-way word-interleaved
L2 Block size	64B
I and D TLB	128 entries, 8KB page size
Memory latency	300 cycles for the first chunk
Network topology	Grid
Flow control mechanism	Virtual channel
No. of virtual channels	4 /physical channel
Back pressure handling	Credit based flow control

appropriate contention values in the look-up table are taken into account during the design space exploration. Some of this empirical data is represented in Figure 3.9. It is observed that for many-core systems, the contention in the network can be as high as 30 cycles per access (for a two banked model) and cannot be ignored during the design space exploration.

For a network with completely pipelined links and routers, contention values are only a function of the router topology and bank cycle time and will not be affected by process technology or L2 cache size. It is assumed here that the cache is organized as static-NUCA (SNUCA), where the address index bits determine the unique bank where the address can be found and the access distribution does not vary greatly as a function of the cache size. CACTI is designed to be more generic than specific. The contention values are provided as a guideline to most users. If a user is interested in a more specific NUCA policy, there is no substitute to generating the corresponding contention values and incorporating them in the tool.

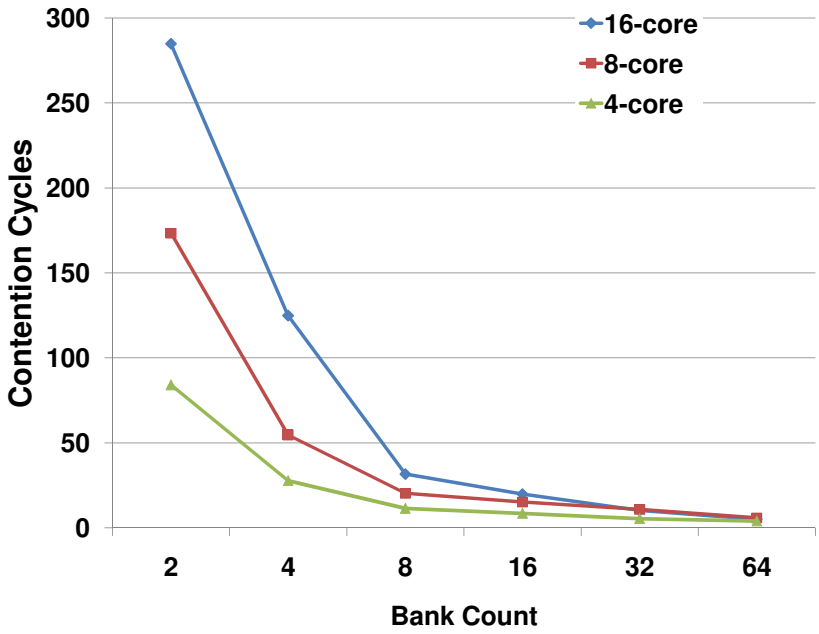


Figure 3.9. Total network contention value/access for CMPs with different NUCA organizations

As a case study in Section 3.3, a different NUCA policy is examined. If CACTI is being employed to compute an optimal L3 cache organization, the contention values will likely be much less because the L2 cache filters out most requests. To handle this case, the average contention values are also computed assuming a large 2 MB L1 cache and this is incorporated into the model as well. In summary, the network contention values are impacted by the following parameters: M, N, bank cycle time, number of cores, router configuration (VCs, buffers), and size of preceding cache.

Figure 3.10 shows an example design space exploration for a 32 MB NUCA L2 cache while attempting to minimize latency. The X-axis shows the number of banks into which the cache is partitioned. For each point on the X-axis, many different bank organizations are considered and the organization with optimal delay (averaged across all banks) is finally represented on the graph. The Y-axis represents this optimal delay and it is further broken down to represent the contributing components: bank access time, link and router delay, router and bank contention.

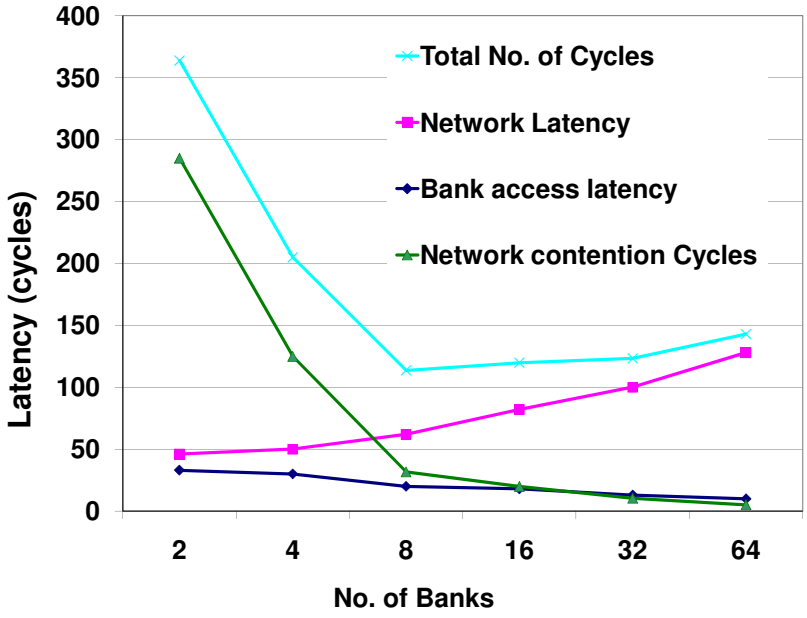


Figure 3.10. NUCA design space exploration

It is observed that the optimal delay is experienced when the cache is organized as a 2×4 grid of 8 banks. The improved methodology significantly impacts the processor performance and power consumption. On an average, the new cache model improves performance by 114% along with 50% reduction in cache access power. See the next chapter for methodology and detailed graphs.

3.2.3 Router Models

The ubiquitous adoption of the system-on-chip (SoC) paradigm and the need for high bandwidth communication links between different modules have led to a number of interesting proposals targeting high-speed network switches/routers [40, 88, 90, 98, 99]. This section provides a brief overview of router complexity and different pipelining options available. It ends with a summary of the delay and power assumptions made for our NUCA CACTI model. For all the evaluations, virtual channel flow control is assumed because of its high throughput and ability to avoid deadlock in the network [40].

Figure 3.11 shows a typical virtual channel router architecture and Figure 3.12 shows the different steps involved in routing a message to the appropriate destination [42]. A flit is the smallest unit of flow control and is usually the number of bits transmitted on a link in a single cycle. The size of the message sent through the network is measured in terms of flits. Every network message consists of a head flit that carries details about the destination of the message and a tail flit indicating the end of the message. If the message size is very small, the head flit can also serve the tail flit's functionality. The highlighted blocks in Figure 3.12 correspond to stages that are specific to head flits. Whenever a head flit of a new message arrives at an input port, the router stores the message in the input buffer and the input controller decodes the message to find the destination. After the decode process, it is then fed to a virtual channel (VC) allocator. The VC allocator consists of a set of arbiters and control logic that takes in requests from messages in all the input ports and allocates appropriate output virtual channels at the destination. If two head flits compete for the same channel, then depending on the priority set in the arbiter, one of the flits gains control of the VC. Upon successful allocation of the VC, the head flit proceeds to the switch allocator. Once the decoding and VC allocation of the head flit are completed, the remaining flits perform nothing in the first two stages. The switch allocator reserves the crossbar so the flits can be forwarded to the appropriate output port. Finally, after the entire message is handled, the tail flit de-allocates the VC. Thus, a typical router pipeline consists of four different stages with the first two stages playing a role only for head flits.

Peh et al. [98] propose a speculative router model to reduce the pipeline depth of virtual channel routers. In their pipeline, switch allocation happens speculatively, in parallel with VC allocation. If the VC allocation is not successful, the message is prevented from entering the final stage, thereby wasting the reserved crossbar time slot. To avoid performance penalty due to mis-speculation, the switch arbitration gives priority to nonspeculative requests over speculative ones. This new model implements the router as a three-stage pipeline.

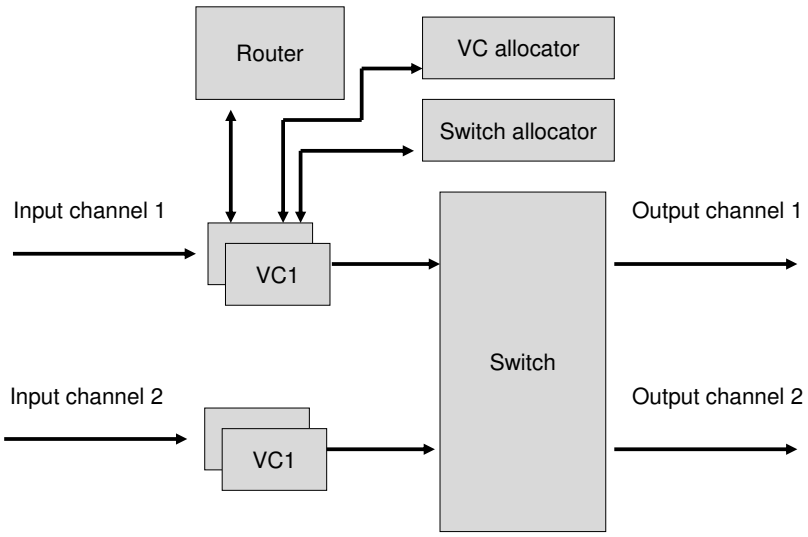


Figure 3.11. Router architecture

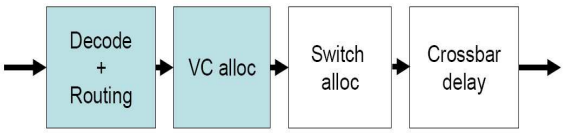


Figure 3.12. Router pipeline [42]

The bulk of the delay in router pipeline stages comes from arbitration and other control overheads. Mullins et al. [90] remove the arbitration overhead from the critical path by precomputing the grant signals. The arbitration logic precomputes the grant signal based on requests in previous cycles. If there are no requests present in the previous cycle, one viable option is to speculatively grant permission to all the requests. If two conflicting requests get access to the same channel, one of the operations is aborted. While successful speculations result in a single-stage router pipeline, mis-speculations are expensive in terms of delay and power.

Single stage router pipelines are not yet a commercial reality. At the other end of the spectrum is the high speed 1.2 GHz router in the Alpha 21364 [88]. The router has eight input ports and seven output ports that includes four external ports to connect off-chip components. The router is deeply pipelined with eight pipeline stages (including special stages for wire delay and ECC) to allow the router to run at the same speed as the main core.

Essentially, innovations in router microarchitectures are on-going. The pipeline depth ranges from a single cycle speculative model [90] to an eight stage model in the Alpha 21364 [88]. For the purpose of the study, the moderately aggressive implementation with a 3-stage pipeline [98] is adopted. The power model is also derived from a corresponding analysis by some of the same authors [120]. As a sensitivity analysis, the results also show the effect of employing router microarchitectures with different pipeline latencies.

As discussed earlier, various routers have been proposed with differing levels of speculation and pipeline stages [42, 90, 98]. The number of stages for each router is left as a user-specified input, defaulting to 3 cycles. For router power, the analytical power models for crossbars is employed and arbiters employed in the Orion toolkit [123]. CACTI's RAM model is employed for router buffer power. These represent the primary contributors to network power (in addition to link power, which was discussed in the previous subsection). A grid topology is modelled where each router has 5 inputs and 5 outputs, and considers three points on the power-performance trade-off curve. Each point provides a different number of buffers per virtual channel and a different number of virtual channels per physical channel. Accordingly, a significant variation in buffer capacity (and power) and contention cycles at routers is seen. As before, the contention cycles are computed with detailed network simulations. Table 3.3 specifies the three types of routers and their corresponding buffer, crossbar, and arbiter energy values.

Table 3.3. Energy consumed (in J) by arbiters, buffers, and crossbars for various router configurations at 32nm technology (flit size of 128 bits).

Component	Configuration 1 4 VCs/PC 16 buffers/VC	Configuration 2 2VCs/PC 8 buffers/VC	Configuration 3 2 VCs/PC 2 buffers/VC
Arbiter	0.33e-12	0.27e-12	0.27e-12
Crossbar (avg)	0.99e-11	0.99e-11	0.99e-11
Buffer read operation/VC	0.11e-11	0.76e-12	0.50e-12
Write buffer operation/VC	0.14e-11	0.10e-11	0.82e-12

3.2.4 Improvement in Trade-Off Analysis

For architectural studies, especially those related to memory hierarchy design, an early estimate of cache access time and power for a given input configuration is crucial to make informed decisions. As described in Section 3.2, CACTI 5.0 carries out a design space exploration over various subarray partitions; it then eliminates organizations that have an area that is 50% higher than the optimal area; it further eliminates those organizations that have an access time value more than 10% the minimum value; and finally, it selects an organization using a cost function that minimizes power and cycle time.

Modern processor design is not singularly focused on performance and many designers are willing to compromise some performance for improved power. Many future studies will likely carry out trade-off analyses involving performance, power, and area. To facilitate such analyses, the new version of the tool adopts the following cost function to evaluate a cache organization (taking into account delay, leakage power, dynamic power, cycle time, and area):

$$
\begin{aligned}
cost = & \\
& W_{acc\_time}\frac{acc\_time}{min\_acc\_time} + \\
& W_{dyn\_power}\frac{dyn\_power}{min\_dyn\_power} + \\
& W_{leak\_power}\frac{leak\_power}{min\_leak\_power} + \\
& W_{cycle\_time}\frac{cycle\_time}{min\_cycle\_time} +
\end{aligned}
$$

$$W_{area}\frac{area}{min\_area}$$

The weights for each term ($W_{acc\_time}, W_{dyn\_power}, W_{leak\_power}, W_{cycle\_time}, W_{area}$) indicate the relative importance of each term and these are specified by the user as input parameters in the configuration file:

```
-weight 100 20 20 10 10
```

The above default weights used by the tool reflect the priority of these metrics in a typical modern design. In addition, the following default line in the input parameters specifies the user's willingness to deviate from the optimal set of metrics:

```
-deviate  1000 1000 1000 1000 1000
```

The above line dictates that we are willing to consider a cache organization where each metric, say the access time, deviates from the lowest possible access time by 1000%. Hence, this default set of input parameters specifies a largely unconstrained search space. The following input lines restrict the tool to identify a cache organization that yields least power while giving up at most 10% performance:

```
-weight   0  100 100  0  0
-deviate 10  1000 1000 1000 1000
```

3.3 Case Study

It is expected that CACTI 6.0 will continue to be used in architectural evaluations in many traditional ways: it is often used to estimate cache parameters while setting up architectural simulators. The new API makes it easier for users to make power, delay, and area trade-offs and this feature is expected to be heavily used for architectural evaluations that focus on power-efficiency or are attempting to allocate power/area budgets to cache or processing. With many recent research proposals focused on NUCA organizations, it is also expected the tool to be heavily used in that context. Since it is difficult to generalize NUCA implementations, it is expected that users modeling NUCA designs may need to modify the model's

parameters and details to accurately reflect their NUCA implementation. Hence, as a case study of the tool's operation, an example NUCA evaluation and its interplay with CACTI 6.0 is presented.

Many recent NUCA papers have attempted to improve average cache access time by moving heavily accessed data to banks that are in close proximity to the core [21, 34, 57, 59, 65]. This is commonly referred to as dynamic-NUCA or D-NUCA because a block is no longer mapped to a unique bank and can move between banks during its L2 lifetime. A novel idea is postulated first and then shown how CACTI 6.0 can be employed to evaluate that idea. Evaluating and justifying such an idea could constitute an entire paper – a high-level evaluation that highlights the changes required to CACTI 6.0 is simply focussed here.

For a D-NUCA organization, most requests will be serviced by banks that are close to the cache controller. Further, with D-NUCA, it is possible that initial banks will have to be searched first and the request forwarded on if the data is not found. All of this implies that initial banks see much higher activity than distant banks. To reduce the power consumption of the NUCA cache, it is proposed that heterogeneous banks be employed: the initial banks can employ smaller power-efficient banks while the distant banks can employ larger banks.

For the case study evaluation, it is focussed on a grid-based NUCA cache adjacent to a single core. The ways of a set are distributed across the banks, so a given address may reside in one of many possible banks depending on the way it is assigned to. Similar to D-NUCA proposals in prior work [65], when a block is brought into the cache, it is placed in the most distant way and it is gradually migrated close to the cache controller with a swap between adjacent ways on every access. While looking for data, each candidate bank is sequentially looked up until the data is found or a miss is signaled.

Recall that CACTI 6.0 assumes an S-NUCA organization where sets are distributed among banks and each address maps to a unique bank. When estimating average access time during the design space exploration, it is assumed that each bank is accessed with an equal probability. The network and bank contention values

are also estimated for an S-NUCA organization. Thus, two changes have to be made to the tool to reflect the proposed implementation:

- The design space exploration must partition the cache space into two: the bank sizes for each partition are estimated independently, allowing the initial banks to have one size and the other banks to have a different size.
- Architectural evaluations have to be performed to estimate the access frequencies for each bank and contention values so that average access time can be accurately computed.

With the simulation infrastructure, it is considered a 32 MB 16-way set-associative L2 cache and modeled the migration of blocks across ways as in the above D-NUCA policy. Based on this, the access frequency shown in Figure 3.13 was computed, with many more accesses to initial banks (unlike the S-NUCA case where the accesses per bank are uniform). With this data integrated into CACTI 6.0, the design space exploration loop of CACTI 6.0 was wrapped around with the following loop structure:

```
for i = 0 to 100
  # Assume i% of the cache has one
  # bank size and the remaining
  # (100-i)% has a different bank size
  for the first i% of cache,
      perform CACTI 6.0 exploration
      (with new access frequencies
       and contention)
  for the remaining (100-i)% of cache,
      perform CACTI 6.0 exploration
      (with new access frequencies
      and contention)
```

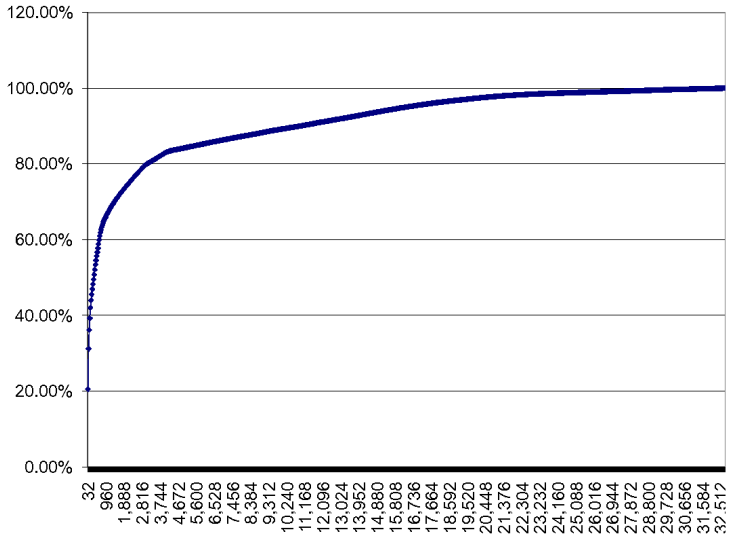


Figure 3.13. Access frequency for a 32MB cache. The y-coordinate of a point in the curve corresponds to the percentage of accesses that can be satisfied with x KB of cache

As an input, the `-weight` and `-deviate` parameters are provided to specify that an organization that minimizes power while yielding performance within 10% of optimal is looked for. The output from this modified CACTI 6.0 indicates that the optimal organization employs a bank size of 4MB for the first 16MB of the cache and a bank size of 8MB for the remaining 16MB. The average power consumption for this organization is 20% lower than the average power per access for the S-NUCA organization yielded by unmodified CACTI 6.0.

3.4 Validation

The newly added modules in CACTI 6.0 are validated against high fidelity SPICE models. This includes low-swing wires, router components, and improved bitline and wordline models. Since SPICE results depend on the model files for transistors, the technology modeling changes made to the recent version of CACTI (version 5) is first discussed followed by the methodology for validating the newly added components to CACTI 6.0.

Earlier versions of CACTI (version one through four) assumed linear technology scaling for calculating cache parameters. All the power, delay, and area values are first calculated for 800nm technology and the results are linearly scaled to the user specified process value. While this approach is reasonably accurate for old process technologies, it can introduce nontrivial error for deep submicron technologies (less than 90nm). This problem is fixed in CACTI 5 [116] by adopting ITRS parameters for all calculations. The current version of CACTI supports four different process technologies (90nm, 65nm, 45nm, and 32nm) with process specific values obtained from ITRS. Though ITRS projections are invaluable for quick analytical estimates, SPICE validation requires technology model files with greater detail and ITRS values cannot be directly plugged in for SPICE verification. The only noncommercial data available publicly for this purpose for recent process technologies is the Predictive Technology Model (PTM) [6]. For our validation, the HSPICE tool was employed along with the PTM 65 nm model file for validating the newly added components. The simulated values obtained from HSPICE are compared against CACTI 6.0 analytical models that take PTM parameters as input [1]. The analytical delay and power calculations performed by the tool primarily depend on the resistance and capacitance parasitics of transistors. For our validation, the capacitance values of source, drain, and gate of n and p transistors are derived from the PTM technology model file. The threshold voltage and the on-resistance of the transistors are calculated using SPICE simulations. In addition to modeling the gate delay and wire delay of different components, our analytical model also considers the delay penalty incurred due to the finite rise time and fall time of an input signal [126].

Figures 3.14 and 3.15 show the comparison of delay and power values of the differential, low-swing analytical models against SPICE values. As mentioned earlier, a low-swing wire model can be broken into three components: transmitter

[1]The PTM parameters employed for verification can be directly used for CACTI simulations. Since most architectural and circuit studies rely on ITRS parameters, CACTI by default assumes ITRS values to maintain consistency.

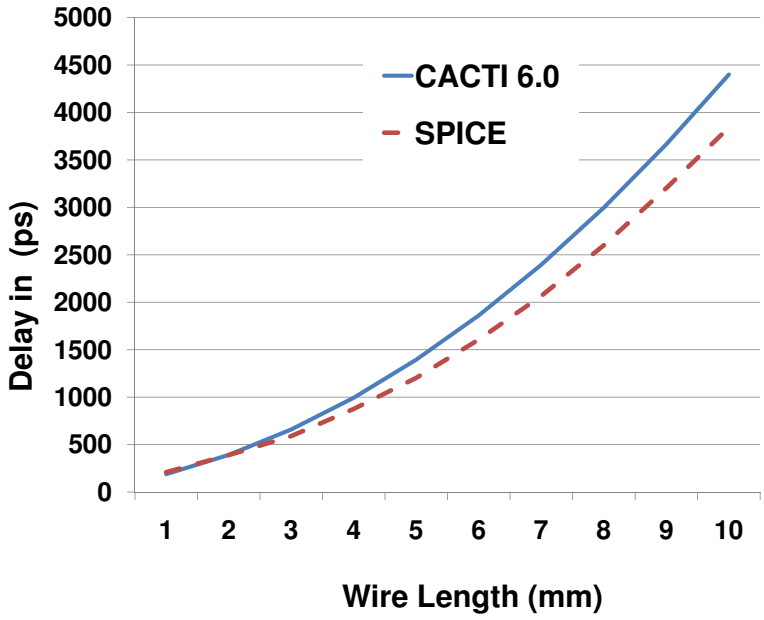


Figure 3.14. Low-swing model delay verification

(that generates the low-swing signal), differential wires[2], and sense amplifiers. The modeling details of each of these components are discussed in Section 2.3.2. Table 3.4 shows the delay and power values of each of these components for a 5mm low-swing wire.

Though the analytical model employed in CACTI 6.0 dynamically calculates the driver size appropriate for a given wire length, for the wire length of our interest, it ends up using the maximum driver size (which is set to 100 times the minimum transistor size) to incur minimum delay overhead. Earlier versions of CACTI also had the problem of overestimating the delay and power values of the sense-amplifier. CACTI 6.0 eliminates this problem by directly using the SPICE generated values for sense-amp power and delay. On an average, the low-swing wire models are verified to be within 12% of the SPICE values.

[2]Delay and power values of low-swing driver is also reported as part of differential wires.

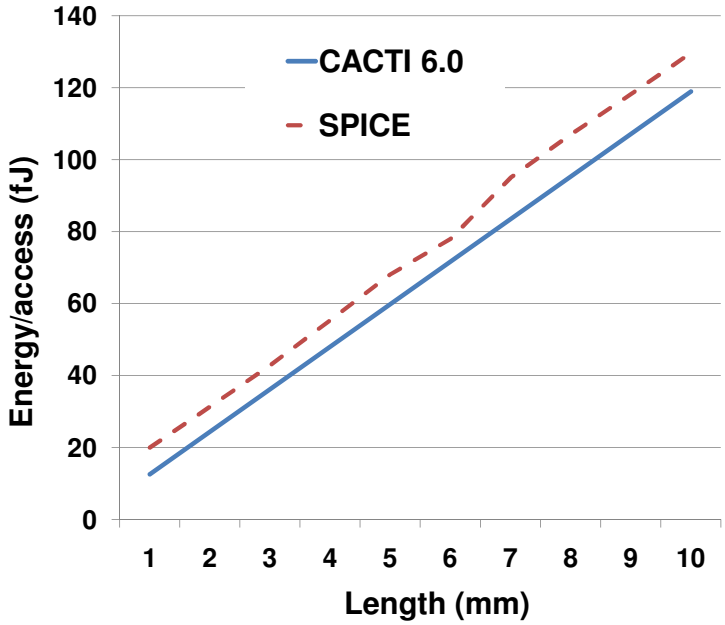


Figure 3.15. Low-swing model energy verification

Table 3.4. Delay and energy values of different components for a 5mm low-swing wire.

Transmitter	Delay, SPICE - 84ps, CACTI - 92ps	Power, SPICE - 7.2fJ , CACTI - 7.5fJ
Differential Wires	Delay, SPICE - 1.204ns, CACTI - 1.395ns	Power, SPICE - 29.9fJ, CACTI - 34fJ
Sense amplifier	Delay, SPICE - 200ps	Power, SPICE - 5.7fJ

The lumped RC model used in prior versions of CACTI for bitlines and wordlines are replaced with a more accurate distributed RC model in CACTI 6.0. Based on a detailed spice modeling of the bitline segment along with the memory cells, the difference between the old and new model is found to be around 11% at 130 nm technology. This difference can go up to 50% with shrinking process technologies as wire parasitics become the dominant factor compared to transistor capacitance [100]. Figure 3.16 and 3.17 compare the distributed wordline and bitline delay values and the SPICE values. The length of the wordlines or bitlines (specified in terms of memory array size) are carefully picked to represent a wide range of cache

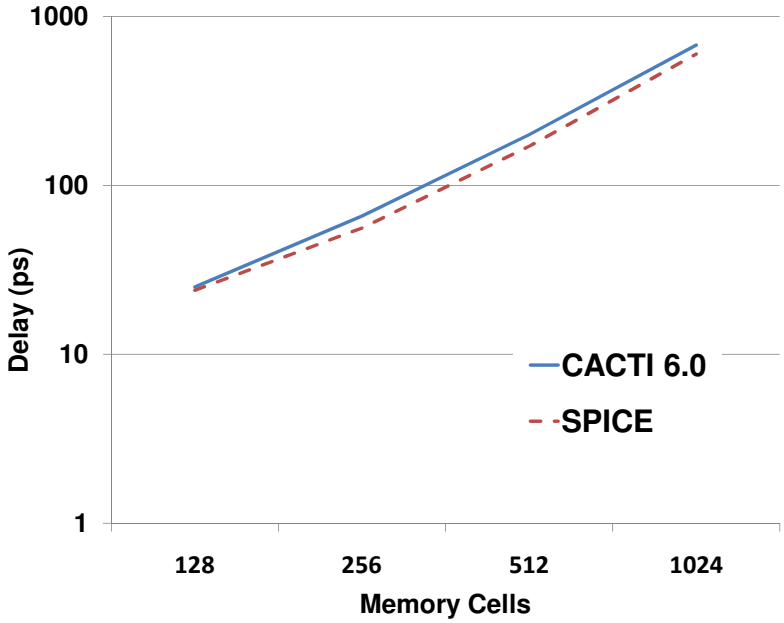


Figure 3.16. Distributed wordline model verification

sizes. On an average, the new analytical models for the distributed wordlines and bitlines are verified to be within 13% and 12% of SPICE generated values.

Buffers, crossbars, and arbiters are the primary components in a router. CACTI 6.0 uses its scratch RAM model to calculate read/write power for router buffers. Orion's arbiter and crossbar model is employed for calculating router power and these models have been validated by Wang et al. [121].

3.5 Related Work

The CACTI tool was first released by Wilton and Jouppi in 1993 [126] and it has undergone four major revisions since then [104, 108, 112]. More details on the latest CACTI 5.0 version are provided in Section 3.2. The primary enhancements of CACTI 2.0 [104] were power models and multi-ported caches; CACTI 3.0 [108] added area models, independently addressed banks, and better sense-amp circuits; CACTI 4.0 [112] improved upon various SRAM circuit structures, moved from aluminium wiring to copper, and included leakage power models; CACTI 5.0 adds

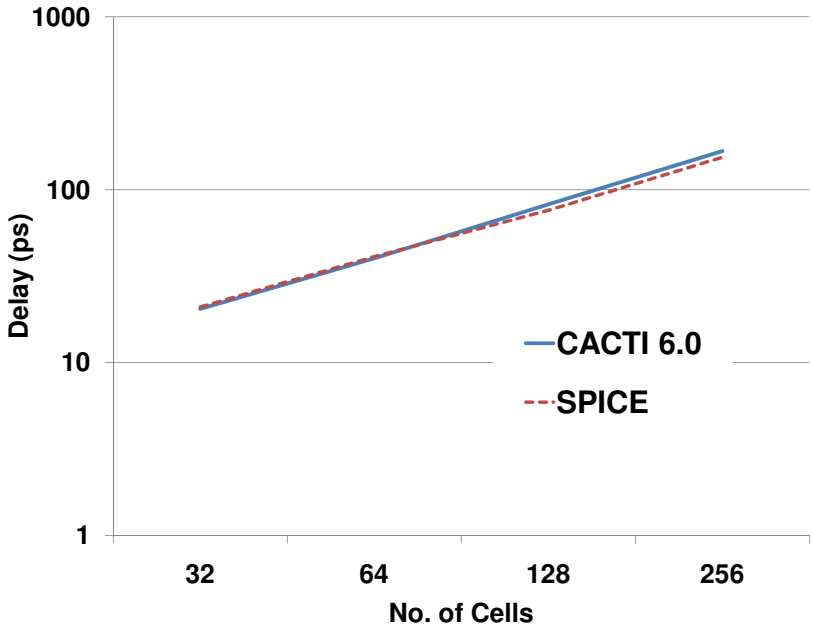


Figure 3.17. Distributed bitline model verification

support for DRAM modeling. The CACTI 6.0 extensions described in this paper represent a major shift in focus and add support for new interconnect components that dominate cache delay and power. Unlike the prior revisions of CACTI that focused on bank and SRAM cell modeling, the current revision focuses on interconnect design. The results in later sections demonstrate that the estimates of CACTI 6.0 are a significant improvement over the estimates of CACTI 5.0.

A few other extensions of CACTI can also be found in the literature, including multiple different versions of *e-CACTI* (enhanced CACTI). eCACTI from the University of California-Irvine models leakage parameters and gate capacitances within a bank in more detail [81] (some of this is now part of CACTI 4.0 [112]). A prior version of eCACTI [3] has been incorporated into CACTI 3.0 [108]. 3DCACTI is a tool that implements a cache across multiple stacked dies and considers the effects of various inter-die connections [117].

A number of tools [10, 27, 44, 51, 111, 119, 123] exist in the literature to model network-on-chip (NoC). The Orion toolkit from Princeton does a thorough

analytical quantification of dynamic and leakage power within router elements [123] and some of this is included in CACTI 6.0. However, Orion does not consider interconnect options, nor carry out a design space exploration (since it is oblivious of the properties of the components that the network is connecting). NOCIC [119] is another model that is based on Spice simulations of various signaling strategies. Given a tile size, it identifies the delay and area needs of each signaling strategy.

3.6 Summary

This chapter describes a novel cache modeling methodology and demonstrates the role of interconnect in deciding the characteristics of large caches. The proposed innovations are incorporated into an open-source cache modeling tool called CACTI. The following points summarize some of the key contributions made to the tool:

- Extends the design space exploration to different wire and router types.
- Considers the use of low-swing differential signaling in addition to traditional global wires.
- Incorporates the effect of network contention during the design space exploration.
- Takes bank cycle time into account in estimating the cache bandwidth.
- Validates a subset of the newly incorporated models.
- Improves upon the tool API, including the ability to specify novel metrics involving power, delay, area, and bandwidth.

The new version of the tool released as CACTI 6.0 identifies a number of relevant design choices on the power-delay-area curves. The estimates of CACTI 6.0 can differ from the estimates of CACTI 5.0 significantly, especially when more fully exploring the power-delay trade-off space. CACTI 6.0 is able to identify cache configurations that can reduce power by a factor of three, while incurring a 25%

delay penalty. Components of the tool are validated against Spice simulations and results showed good agreement between analytical and transistor-level models. Finally, an example case study of heterogeneous NUCA banks is presented that demonstrates how the tool can benefit architectural evaluations.

CHAPTER 4

NOVEL CACHE PIPELINES

4.1 Leveraging Interconnect Choices for Performance Optimizations

The previous chapter discussed the critical role played by interconnects in deciding the power and performance characteristics of large caches. With wire delay accounting for more than 80% of the access time in large caches, a number of innovations are possible by focusing on interconnect design. This chapter discusses how to leverage the high-speed L-wires 2 to improve cache performance. All the proposed innovations guarantee equal complexity with respect to metal area in both baseline model and proposed models.

Consistent with most modern implementations, it is assumed that each cache bank stores the tag and data arrays and that all the ways of a set are stored in a single cache bank. For most of this discussion, it is assumed that there is enough metal area to support a baseline inter-bank network that accommodates 256 data wires and 64 address wires, all implemented as minimum-width wires on the 8X metal plane (the 8X-B-wires in Table 4.1). A higher bandwidth inter-bank network does not significantly improve IPC, so it is believed that this is a reasonable baseline. Next, optimizations that incorporate different types of wires, without exceeding the above metal area budget, is considered.

4.1.1 Early Look-Up

L-wires can be leveraged for low latency, but they consume eight times the area of a B-wire on the 8X metal plane. The implementation of a 16-bit L-network will

Table 4.1. Delay and area characteristics of different wire implementations at 65 nm technology.

Wire Type	Relative Latency	Relative Area	Latency (ns/mm)	Wiring Pitch (nm)
8X-B-Wire	$1x$	$1x$	0.122	210
4X-B-Wire	$2.0x$	$0.5x$	0.244	105
L-Wire	$0.25x$	$8x$	0.03	1680

require that 128 B-wires be eliminated to maintain constant metal area. Consider the following heterogeneous network that has the same metal area as the baseline: 128 B-wires for the data network, 64 B-wires for the address network, and 16 additional L-wires.

In a typical cache implementation, the cache controller sends the complete address as a single message to the cache bank. After the message reaches the cache bank, it starts the look-up and selects the appropriate set. The tags of each block in the set are compared against the requested address to identify the single block that is returned to the cache controller. It is observed that the least significant bits of the address (LSB) are on the critical path because they are required to index into the cache bank and select candidate blocks. The most significant bits (MSB) are less critical since they are required only at the tag comparison stage that happens later. This opportunity can be exploited to break the traditional sequential access. A partial address consisting of LSB can be transmitted on the low bandwidth L-network and cache access can be initiated as soon as these bits arrive at the destination cache bank. In parallel with the bank access, the entire address of the block is transmitted on the slower address network composed of B-wires. (This design choice is referred to as *option-A*). When the entire address arrives at the bank and when the set has been read out of the cache, the MSB is used to select at most a single cache block among the candidate blocks. The data block is then returned to the cache controller on the 128-bit wide data network. The proposed optimization is targeted only for cache reads. Cache writes are not done speculatively and wait for the complete address to update the cache line.

For a 512 KB cache bank with a block size of 64 bytes and a set associativity of 8, only 10 index bits are required to read a set out of the cache bank. Hence, the 16-bit L-network is wide enough to accommodate the index bits and additional control signals (such as destination bank). In terms of implementation details, the coordination between the address transfers on the L-network and the slower address network can be achieved in the following manner. Only a single early look-up is allowed to happen at a time and the corresponding index bits are maintained in a register. If an early look-up is initiated, the cache bank pipeline proceeds just as in the base case until it arrives at the tag comparison stage. At this point, the pipeline is stalled until the entire address arrives on the slower address network. When this address arrives, it checks to see if the index bits match the index bits for the early look-up currently in progress. If the match is successful, the pipeline proceeds with tag comparison. If the match is unsuccessful, the early look-up is squashed and the entire address that just arrived on the slow network is used to start a new L2 access from scratch. Thus, an early look-up is wasted if a different address request arrives at a cache bank between the arrival of the LSB on the L-network and the entire address on the slower address network. If another early look-up request arrives while an early look-up is in progress, the request is simply buffered (potentially at intermediate routers). For the simulations, supporting multiple simultaneous early look-ups was not worth the complexity.

The early look-up mechanism also introduces some redundancy in the system. There is no problem if an early look-up fails for whatever reason – the entire address can always be used to look up the cache. Hence, the transmission on the L-network does not require ECC or parity bits.

Apart from the network delay component, the major contributors to the access latency of a cache are delay due to decoders, wordlines, bitlines, comparators, and drivers. Of the total access time of the cache, depending on the size of the cache bank, around 60-80% of the time has elapsed by the time the candidate sets are read out of the appropriate cache bank. By breaking the sequential access as described above, much of the latency for decoders, bitlines, wordlines, etc., is hidden

behind network latency. In fact, with this optimization, it may even be possible to increase the size of a cache bank without impacting overall access time. Such an approach will help reduce the number of network routers and their corresponding power/area overheads. In an alternative approach, circuit/VLSI techniques can be used to design banks that are slower and consume less power (for example, the use of body-biasing and high-threshold transistors). The exploration of these optimizations is left for future work.

4.1.2 Aggressive Look-Up

While the previous technique is effective in hiding a major part of the cache access time, it still suffers from long network delays in the transmission of the entire address over the B-wire network. In an alternative implementation (referred to as *option-B*), the 64-bit address network can be eliminated and the entire address is sent in a pipelined manner over the 16-bit L-network. Four flits are used to transmit the address, with the first flit containing the index bits and initiating the early look-up process. In Section 6.4, it is shown that this approach increases contention in the address network and yields little performance benefit.

To reduce the contention in the L-network, an optimization is introduced that is referred to as *Aggressive look-up* (or *option-C*). By eliminating the 64-bit address network, the width of the L-network can be increased by eight bits without exceeding the metal area budget. Thus, in a single flit on the L-network, in addition to the index bits required for an early look-up, eight bits of the tag is also transmitted. For cache reads, the rest of the tag is not transmitted on the network. This subset of the tag is used to implement a partial tag comparison at the cache bank. Cache writes still require the complete address and the address is sent in multiple flits over the L-network. According to this simulations, for 99% of all cache reads, the partial tag comparison yields a single correct matching data block. In the remaining cases, false positives are also flagged. All blocks that flag a partial tag match must now be transmitted back to the CPU cache controller (along with their tags) to implement a full tag comparison and locate the required data. Thus, the bandwidth demands

are reduced on the address network at the cost of higher bandwidth demands on the data network. As shown in the results, this is a worthwhile trade-off.

With the early look-up optimization, multiple early look-ups at a bank are disallowed to simplify the task of coordinating the transmissions on the L and B networks. The aggressive look-up optimization does not require this coordination, so multiple aggressive look-ups can proceed simultaneously at a bank. On the other hand, ECC or parity bits are now required for the L-network because there is no B-network transmission to fall back upon in case of error. The L-network need not accommodate the MSHR-id as the returned data block is accompanied with the full tag. In a CMP, the L-network must also include a few bits to indicate where the block must be sent to. Partial tag comparisons exhibit good accuracy even if only five tag bits are used, so the entire address request may still fit in a single flit. The probability of false matches can be further reduced by performing tag transformation and carefully picking the partial tag bits [64].

In a CMP model that maintains coherence among L1 caches, depending on the directory implementation, aggressive look-up will attempt to update the directory state speculatively. If the directory state is maintained at cache banks, aggressive look-up may eagerly update the directory state on a partial tag match. Such a directory does not compromise correctness, but causes some unnecessary invalidation traffic due to false positives. If the directory is maintained at a centralized cache controller, it can be updated nonspeculatively after performing the full tag-match.

Clearly, depending on the bandwidth needs of the application and the available metal area, any one of the three discussed design options may perform best. The point here is that the choice of interconnect can have a major impact on cache access times and is an important consideration in determining an optimal cache organization. Given the set of assumptions, the results in the next section show that option-C performs best, followed by option-A, followed by option-B.

4.1.3 Hybrid Network

The optimal cache organization selected by CACTI 6.0 is based on the assumption that each link employs B-wires for data and address transfers. The discussion in the previous two subsections makes the case that different types of wires in the address and data networks can improve performance. If L-wires are employed for the address network, it often takes less than a cycle to transmit a signal between routers. Therefore, part of the cycle time is wasted and most of the address network delay is attributed to router delay. Hence, an alternative topology for the address network is proposed. By employing fewer routers, full advantage of the low latency L-network is taken and lowers the overhead from routing delays. The corresponding penalty is that the network supports a lower overall bandwidth.

Figure 4.1 shows the proposed hybrid topology to reduce the routing overhead in the address network for uniprocessor models. The address network is now a combination of point-to-point and bus architectures. Each row of cache banks is allocated a single router and these routers are connected to the cache controllers with a point-to-point network, composed of L-wires. The cache banks in a row share a bus composed of L-wires. When a cache controller receives a request from the CPU, the address is first transmitted on the point-to-point network to the appropriate row and then broadcast on the bus to all the cache banks in the row. Each hop on the point-to-point network takes a single cycle (for the 4x4-bank model) of link latency and three cycles of router latency. The broadcast on the bus does not suffer from router delays and is only a function of link latency (2 cycles for the 4x4 bank model). Since the bus has a single master (the router on that row), there are no arbitration delays involved. If the bus latency is more than a cycle, the bus can be pipelined [69]. For the simulations in this study, it is assumed that the address network is always 24 bits wide (just as in option-C above) and the aggressive look-up policy is adopted (blocks with partial tag matches are sent to the CPU cache controller). As before, the data network continues to employ the grid-based topology and links composed of B-wires (128-bit network, just as in option-C above).

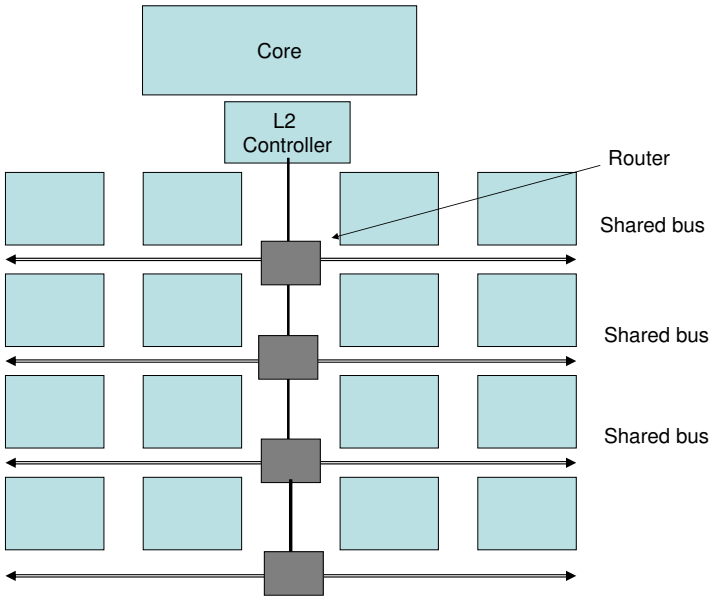


Figure 4.1. Hybrid network topology for a uniprocessor.

A grid-based address network (especially one composed of L-wires) suffers from huge metal area and router overheads. The use of a bus composed of L-wires helps eliminate the metal area and router overhead, but causes an inordinate amount of contention for this shared resource. The hybrid topology that employs multiple buses connected with a point-to-point network strikes a good balance between latency and bandwidth as multiple addresses can simultaneously be serviced on different rows. Thus, in this proposed hybrid model, three forms of heterogeneity have been introduced: (i) different types of wires are being used in data and address networks, (ii) different topologies are being used for data and address networks, (iii) the address network uses different architectures (bus-based and point-to-point) in different parts of the network.

4.2 Results

4.2.1 Methodology

This simulator is based on Simplescalar-3.0 [26] for the Alpha AXP ISA. Table 4.2 summarizes the configuration of the simulated system. All the delay and

Table 4.2. Simplescalar simulator parameters.

Fetch queue size	64
Branch predictor	comb. of bimodal and 2-level
Bimodal predictor size	16K
Level 1 predictor	16K entries, history 12
Level 2 predictor	16K entries
BTB size	16K sets, 2-way
Branch mispredict penalty	at least 12 cycles
Fetch width	8 (across up to 2 basic blocks)
Dispatch and commit width	8
Issue queue size	60 (int and fp, each)
Register file size	100 (int and fp, each)
Re-order Buffer size	80
L1 I-cache	32KB 2-way
L1 D-cache	32KB 2-way set-associative,
L2 cache	32MB 8-way SNUCA 3 cycles, 4-way word-interleaved
L2 Block size	64B
I and D TLB	128 entries, 8KB page size
Memory latency	300 cycles for the first chunk

power calculations are for a 65 nm process technology and a clock frequency of 5 GHz. Contention for memory hierarchy resources (ports and buffers) is modeled in detail. A 32 MB on-chip level-2 static-NUCA cache is assumed and employs a grid network for communication between different L2 banks.

The network employs two unidirectional links between neighboring routers and virtual channel flow control for packet traversal. The router has five input and five output ports. Four virtual channels are assumed for each physical channel and each channel has four buffer entries (since the flit counts of messages are small, four buffers are enough to store an entire message). The network uses adaptive routing similar to the Alpha 21364 network architecture [88]. If there is no contention, a message attempts to reach the destination by first traversing in the horizontal direction and then in the vertical direction. If the message encounters a stall, in the next cycle, the message attempts to change direction, while still attempting to reduce the Manhattan distance to its destination. To avoid deadlock due to adaptive routing, of the four virtual channels associated with each vertical physical link, the fourth virtual channel is used only if a message destination is in that column. In

other words, messages with unfinished horizontal hops are restricted to use only the first three virtual channels. This restriction breaks the circular dependency and provides a safe path for messages to drain via deadlock-free VC4. All the proposals are evaluated for uniprocessor and CMP processor models. This CMP simulator is also based on Simplescalar and employs eight out-of-order cores and a shared 32MB level-2 cache. For most simulations, the same network bandwidth parameters outlined in Section 6.3 is assumed and reiterated in Table 4.3. Since network bandwidth is a bottleneck in the CMP shown, the CMP results with twice as much bandwidth. As a workload, the SPEC2k programs employed is executed for 100 million instruction windows identified by the Simpoint toolkit [107]. The composition of programs in this multiprogrammed CMP workload is described in the next subsection.

4.2.2 IPC Analysis

The behavior of processor models is examined with eight different cache configurations summarized in Table 4.3. The first six models help demonstrate the improvements from the most promising novel designs, and the last two models show results for other design options that were also considered and serve as useful comparison points.

The first model is based on methodologies in prior work [65], where the bank size is calculated such that the link delay across a bank is less than one cycle. All other

Table 4.3. Summary of different models simulated. Global 8X wires are assumed for the inter-bank links. "A" and "D" denote the address and data networks, respectively.

MODEL	LINK LATENCY (VERT,HORIZ)	BANK ACCESS TIME	BANK COUNT	NETWORK LINK CONTENTS	DESCRIPTION
MODEL 1	1,1	3	512	B-WIRES (256D, 64A)	BASED ON PRIOR WORK
MODEL 2	4,3	17	16	B-WIRES (256D, 64A)	DERIVED FROM CACTIL2
MODEL 3	4,3	17	16	B-WIRES (128D, 64A) & L-WIRES (16A)	IMPLEMENTS EARLY LOOK-UP
MODEL 4	4,3	17	16	B-WIRES (128D) & L-WIRES (24A)	IMPLEMENTS AGGRESSIVE LOOK-UP
MODEL 5	HYBRID	17	16	L-WIRES (24A) & B-WIRES (128D)	LATENCY-BANDWIDTH TRADEOFF
MODEL 6	4,3	17	16	B-WIRES (256D), 1CYCLE ADD	IMPLEMENTS OPTIMISTIC CASE
MODEL 7	1,1	17	16	L-WIRES (40A/D)	LATENCY OPTIMIZED
MODEL 8	4,3	17	16	B-WIRES (128D) & L-WIRES (24A)	ADDRESS-L-WIRES & DATA-B-WIRES

models employ the proposed CACTI 6.0 tool to calculate the optimum bank count, bank access latency, and link latencies (vertical and horizontal) for the grid network. Model two is the baseline cache organization obtained with CACTI 6.0 that employs minimum-width wires on the 8X metal plane for the address and data links. Model three and four augment the baseline interconnect with an L-network to accelerate cache access. Model three implements the early look-up proposal (Section 4.1.1) and model four implements the aggressive look-up proposal (Section 4.1.2). Model five simulates the hybrid network (Section 4.1.3) that employs a combination of bus and point-to-point network for address communication. As discussed in Section 6.3, the bandwidths of the links in all these simulated models are adjusted such that the net metal area is constant. All of the above optimizations help speed up the address network and do not attempt to improve the data network. To get an idea of the best performance possible with such optimizations to the address network, an optimistic model (model six) is simulated where the request carrying the address magically reaches the appropriate bank in one cycle. The data transmission back to the cache controller happens on B-wires just as in the other models.

Model seven employs a network composed of only L-wires and both address and data transfers happen on the L-network. Due to the equal metal area restriction, model seven offers lower total bandwidth than the other models and each message is correspondingly broken into more flits. Model eight simulates the case where the address network is entirely composed of L-wires and the data network is entirely composed of B-wires. This is similar to model four, except that instead of performing a partial tag match, this model sends the complete address in multiple flits on the L-network and performs a full tag match.

Figure 4.2 shows the IPCs (average across the SPEC2k suite) for all eight processor models, normalized against model one. It also shows the average across programs in SPEC2k that are sensitive to L2 cache latency. Figure 4.3 shows the IPCs for models one through six for each individual program (L2 sensitive programs are highlighted in the figure). Table 4.4 quantifies the average L2 access times with the proposed optimizations as a function of bank count. In spite of having the least

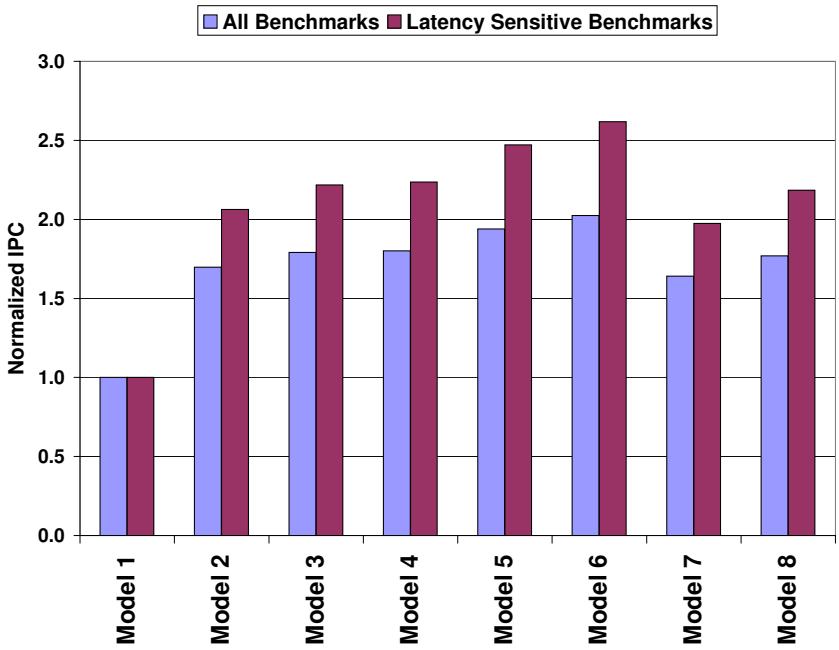


Figure 4.2. Normalized IPCs of SPEC2000 benchmarks for different L2 cache configurations (B-wires implemented on the 8X metal plane).

possible bank access latency (3 cycles as against 17 cycles for other models), model one has the poorest performance due to high network overheads associated with each L2 access. Model two, the performance-optimal cache organization derived from CACTI 6.0, performs significantly better, compared to model one. On an average, model two's performance is 73% better across all the benchmarks and 114% better for benchmarks that are sensitive to L2 latency. This performance improvement is accompanied by reduced power and area from using fewer routers.

The early look-up optimization discussed in Section 4.1.1 improves upon the performance of model two. On an average, model three's performance is 6% better, compared to model two across all the benchmarks and 8% better for L2-sensitive benchmarks. Model four further improves the access time of the cache by performing the early look-up and aggressively sending all the blocks that exhibit partial tag matches. This mechanism has 7% higher performance, compared to model two across all the benchmarks, and 9% for L2-sensitive benchmarks. The low performance improvement of model four is mainly due to the high router overhead

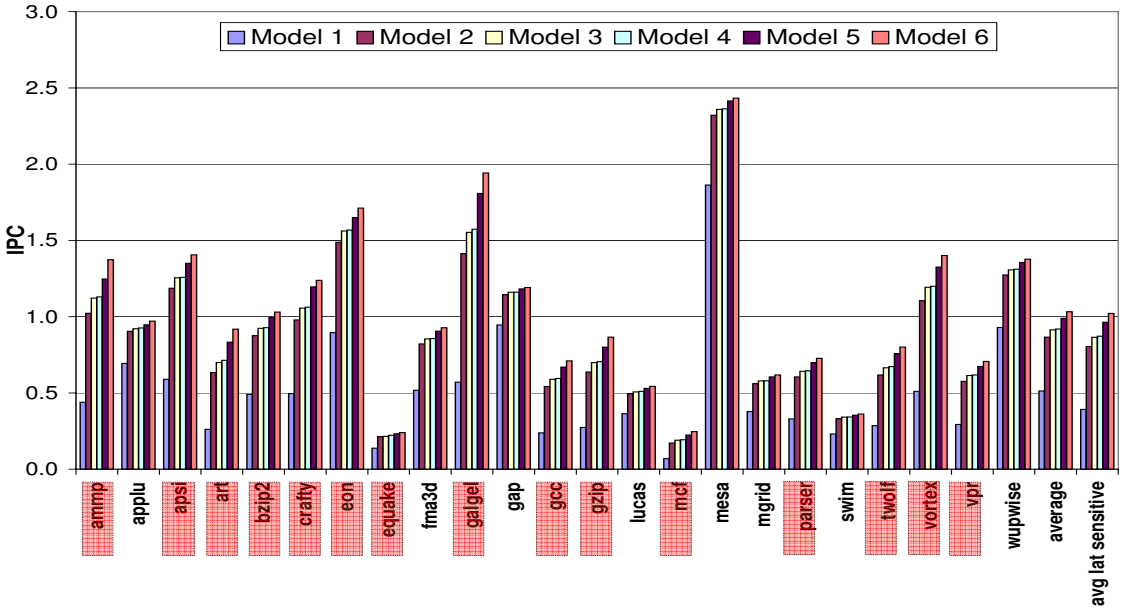


Figure 4.3. IPCs of SPEC2000 benchmarks for different L2 cache configurations (B-wires implemented on the 8X metal plane).

associated with each transfer. The increase in data network traffic from partial tag matches is less than 1%. The aggressive and early look-up mechanisms trade off data network bandwidth for a low-latency address network. By halving the data network's bandwidth, the delay for the pipelined transfer of a cache line increases by two cycles (since the cache line takes up two flits in the baseline data network). This enables a low-latency address network that can save two cycles on every hop, resulting in a net win in terms of overall cache access latency. The narrower data network is also susceptible to more contention cycles, but this was not a major factor for the evaluated processor models and workloads.

The hybrid model overcomes the shortcomings of model four by reducing the number of routers in the network. Aggressive look-up implemented in a hybrid topology (model five) performs the best and is within a few percent of the optimistic model six. Compared to model two, the hybrid model performs 15% better across all benchmarks and 20% better for L2-sensitive benchmarks.

Model seven employs the L-network for transferring both address and data messages. The performance of this model can be better than the optimistic model

Table 4.4. Access latencies for different cache configurations. The operating frequency is 5GHz and the message transfers are assumed to happen on 8x wires.

Bank Count (32 MB Cache)	Bank Access Time (cycles)	Average Cache Access Time (cycles)	Early Look-up (cycles)	Aggressive Fetch (cycles)	Optimistic Model (cycles)
2	77	159	137	137	132
4	62	151	127	127	118
8	26	97	78.5	77.5	66.5
16	17	84	70.5	69.5	54.5
32	9	87	78.7	73.5	50.5
64	6	104	98.1	88	57
128	5	126	121.2	114	67
256	4	148	144.6	140.5	77.5
512	3	226	223.3	211	116.5
1024	3	326	323	293	166.5
2048	3	419	416.2	403.5	212.5
4096	3	581	578	548.5	293

(model six) that uses B-wires for data transfers. But the limited bandwidth of the links in model seven increases contention in the network and limits the performance improvement to only a few programs that have very low network traffic. On an average, the performance of model seven is 4% less than model two. Model eight employs the L-network for sending the complete address in a pipelined fashion. It performs comparably to model four that implements aggressive look-up (4.4% better than model two). However, it incurs significantly higher contention on the L-network, making it an unattractive choice for CMPs.

Figure 4.4 shows the IPC improvement of different models in a CMP environment. All the models are evaluated for four different sets of multiprogrammed workloads. Set 1 is a mixture of benchmarks with different characteristics. Set 2 consists of benchmarks that are sensitive to L2 hit time. Half the programs in set 3 are L2-sensitive and the other half are not. Set 4 consists of benchmarks that are memory intensive. The individual programs in each set are listed in Figure 4.4. For the results, it is assumed that the network bandwidth is doubled (by assuming twice as much metal area) to support the increased demands from eight cores. Similar to the uniprocessor results, model one incurs severe performance penalty due to very high network overhead. Model two, derived from CACTI 6.0, out-performs model one by 51%. Models three, four, and five yield performance improvements

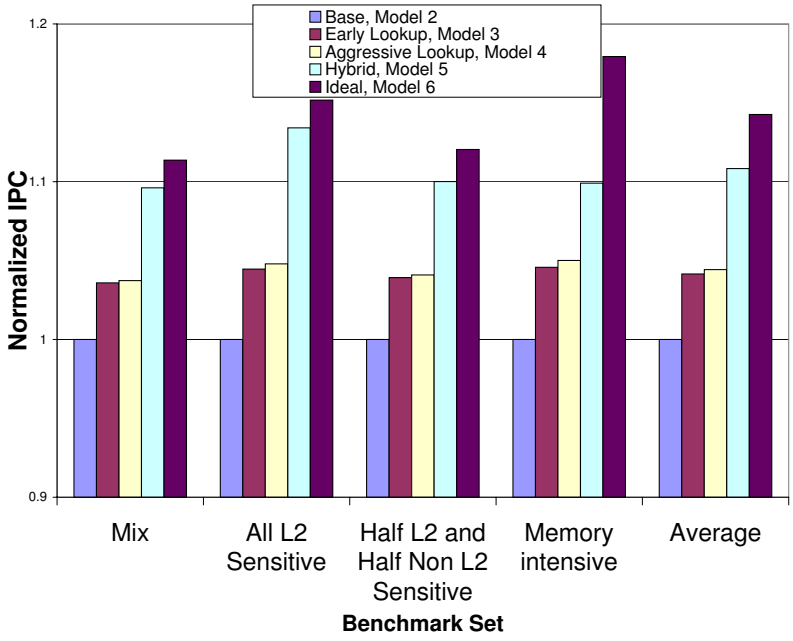


Figure 4.4. IPC improvement of different cache configurations in an eight core CMP. Benchmark compositions: "Mix" - ammp, applu, lucas, bzip2, crafty, mgrid, equake, gcc; "All sensitive" - ammp, apsi, art, bzip2, crafty, eon, equake, gcc; "Half L2 and Half Non-L2 sensitive" - ammp, applu, lucas, bzip2, crafty, mgrid, mesa, gcc; "Memory intensive" - applu, fma3d, art, swim, lucas, equake, gap, vpr.

of 4.2% (early look-up), 4.5% (aggressive look-up), and 10.8% (hybrid network), respectively, over model two. If the network bandwidth is the same as in the uniprocessor simulations, these performance improvements are 3.0%, 3.3%, and 6.2%, respectively.

As a sensitivity analysis, the overall IPC improvements are reported for models two through six for the uniprocessor model with 4X-B wires instead of 8X-B wires in Figure 4.5. Since 4X wires are slower, the effect of optimizations are more pronounced than in the 8X model. This is likely the expected trend in future technologies where wires are slower, relative to logic delays.

4.3 Related Work

A number of recent proposals have dealt with the implementation of large NUCA caches [21, 33, 34, 57, 59, 65] and focus on optimizing logical policies

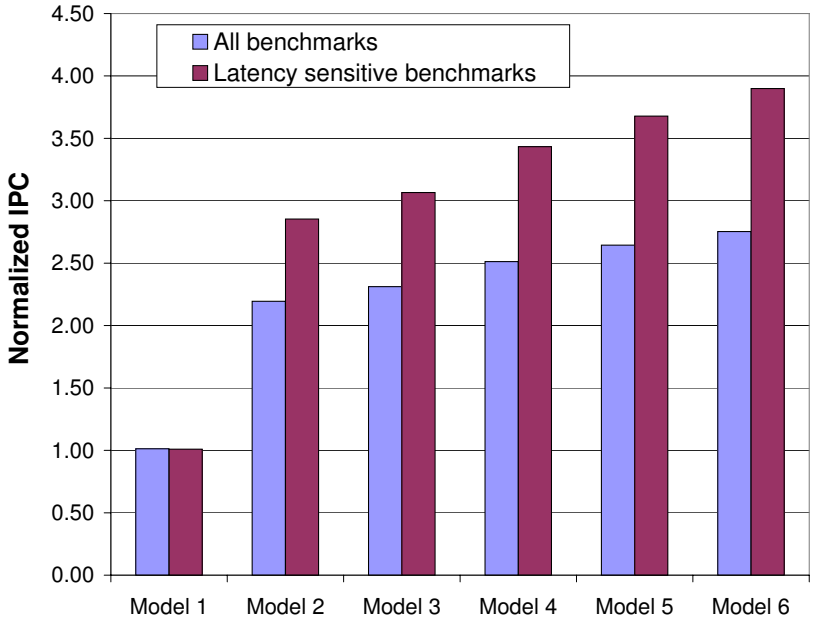


Figure 4.5. Performance for uniprocessor with 4X wires.

associated with a baseline cache design. For example, many of these papers employ some form of dynamic-NUCA (D-NUCA), where blocks are allowed to migrate to reduce communication distances. D-NUCA policies also apply to cache models that incorporate the interconnect optimizations proposed in this work.

To the best of my knowledge, only four other bodies of work have attempted to exploit novel interconnects at the microarchitecture level to accelerate cache access. Beckmann and Wood [20] employ transmission lines to speed up access to large caches. Unlike regular RC-based wires, transmission lines do not need repeaters and hence can be directly routed on top of other structures. This property is exploited to implement transmission line links between each cache bank and the central cache controller. The number of banks is limited to the number of links that can be directly connected to the controller. Low-latency *fat* RC-based wires have been employed to speed up coherence signals in a CMP environment [32] and L1 cache access in a clustered architecture [13]. A recent paper by Li et al. [75] proposes the implementation of a NUCA cache in three dimensions. A three-dimensional grid topology is employed and given the low latency for inter-die communication;

a dynamic time division multiplexing bus is employed for signal broadcast across dies. Jin et al. [59] suggest removing unnecessary links in a grid topology to reduce the area and power overhead of the network. They propose Halo network and a multi-cast router architecture for the D-NUCA domain.

Kumar et al. [69] examine interconnect design issues associated with chip multiprocessors. They detail the scalability problem associated with a shared bus fabric and explore the potential of a hierarchical bus structure. Kumar et al. [68] propose Express Virtual Channels (EVC) to alleviate long pipeline overhead associated with on-chip routers. EVC is a flow control mechanism that exploits special high priority virtual lanes to bypass router pipeline. Messages that require multiple router hops are routed through express lanes to reduce communication delay. EVC's high priority guarantees switch allocation and helps messages bypass regular router arbitration and allocation stages. While their mechanism can reduce average hop latency of a message, employing high priority EVCs can lead to starvation in regular messages. Furthermore, a network with EVCs will continue to incur high link overhead and can benefit from heterogeneous wires to further reduce network overhead.

4.4 Summary

Delays within wires and routers are major components of L2 cache access time. In the previous chapter, a methodology was proposed to obtain an optimal baseline model for large caches. This chapter builds upon the model and discusses novel optimizations to the address network and bank access pipeline that help hide network delays. These optimizations leverage heterogeneity within the network and improve upon the IPC of the baseline by 15% across the SPEC benchmark suite.

This work has focused on the design of a NUCA cache shared by all cores on a chip. Private L2 cache organizations are also being considered by industry and academia – each core is associated with a large L2 cache and a request not found in a local cache may be serviced by a remote cache [19, 30, 34, 85, 109, 131]. A

remote L2 hit now has a nonuniform access time depending on the remote cache where the block is found and the network delays incurred in communication with the directory and the remote cache. The interconnect continues to play a major role in such cache organizations and many of the interconnect design considerations for a shared NUCA will also apply to private L2 caches.

The discussions so far have only exploited low-latency L-wires to improve performance. As described in Chapter 2, wires can be designed to minimize power (while trading off latency). The work can be further extended by considering techniques to leverage power-efficient wires to reduce interconnect power while handling prefetches, writebacks, block swaps, etc. Latency tolerant networks can also enable power optimizations within cache banks (mentioned in Section 4.1.1).

CHAPTER 5

HETEROGENEOUS INTERCONNECTS FOR COHERENCE TRANSACTIONS

The optimizations proposed so far are targeted at accelerating the basic read/write operations to large caches. With the proliferation of Chip-Multiprocessors (CMP), modern cache hierarchies incur an additional overhead due to coherence transactions. This chapter presents a number of techniques by which coherence traffic within a CMP can be mapped intelligently to different wire implementations with minor increases in complexity. Such an approach can not only improve performance, but also reduce power dissipation.

5.1 Coherence Overhead

In a typical CMP, the L2 cache and lower levels of the memory hierarchy are shared by multiple cores [67, 115]. Sharing the L2 cache allows high cache utilization and avoids duplicating cache hardware resources. L1 caches are typically not shared as such an organization entails high communication latencies for every load and store. There are two major mechanisms used to ensure coherence among L1s in a chip multiprocessor. The first option employs a bus connecting all of the L1s and a snoopy bus-based coherence protocol. In this design, every L1 cache miss results in a coherence message being broadcast on the global coherence bus and other L1 caches are responsible for maintaining valid state for their blocks and responding to misses when necessary. The second approach employs a centralized directory in the L2 cache that tracks sharing information for all cache lines in the L2. In such a directory-based protocol, every L1 cache miss is sent to the L2 cache, where further actions are taken based on that block's directory state. Many studies [2, 22, 56, 70, 74] have characterized the high frequency of cache misses

in parallel workloads and the high impact these misses have on total execution time. On a cache miss, a variety of protocol actions are initiated, such as request messages, invalidation messages, intervention messages, data block writebacks, data block transfers, etc. Each of these messages involves on-chip communication with latencies that are projected to grow to tens of cycles in future billion transistor architectures [3].

This chapter explores optimizations that are enabled when a heterogeneous interconnect is employed for coherence traffic. For example, when employing a directory-based protocol, on a cache write miss, the requesting processor may have to wait for data from the home node (a two hop transaction) and for acknowledgments from other sharers of the block (a three hop transaction). Since the acknowledgments are on the critical path and have low bandwidth needs, they can be mapped to wires optimized for delay, while the data block transfer is not on the critical path and can be mapped to wires that are optimized for low power.

5.2 Optimizing Coherence Traffic

For each cache coherence protocol, there exist a variety of coherence operations with different bandwidth and latency needs. Because of this diversity, there are many opportunities to improve performance and power characteristics by employing a heterogeneous interconnect. The goal of this section is to present a comprehensive listing of such opportunities. The protocol-specific optimizations are focused on in Section 5.2.1 and on protocol-independent techniques in Section 5.2.2. The implementation complexity of these techniques is discussed in Section 5.3.

5.2.1 Protocol-Dependent Techniques

The characteristics of operations is first examined in both directory-based and snooping bus-based coherence protocols and how they can map to different sets of wires. In a bus-based protocol, the ability of a cache to directly respond to another cache's request leads to low L1 cache-to-cache miss latencies. L2 cache latencies are relatively high as a processor core has to acquire the bus before sending a

request to the L2. It is difficult to support a large number of processor cores with a single bus due to the bandwidth and electrical limits of a centralized bus [24]. In a directory-based design [37, 72], each L1 connects to the L2 cache through a point-to-point link. This design has low L2 hit latency and scales better. However, each L1 cache-to-cache miss must be forwarded by the L2 cache, which implies high L1 cache-to-cache latencies. The performance comparison between these two design choices depends on the cache sizes, miss rates, number of outstanding memory requests, working-set sizes, sharing behavior of targeted benchmarks, etc. Since either option may be attractive to chip manufacturers, both forms of coherence protocols are considered in this study.

- **Write-Invalidate Directory-Based Protocol**

  Write-invalidate directory-based protocols have been implemented in existing dual-core CMPs [115] and will likely be used in larger scale CMPs as well. In a directory-based protocol, every cache line has a directory where the states of the block in all L1s are stored. Whenever a request misses in an L1 cache, a coherence message is sent to the directory at the L2 to check the cache line's global state. If there is a clean copy in the L2 and the request is a READ, it is served by the L2 cache. Otherwise, another L1 must hold an exclusive copy and the READ request is forwarded to the exclusive owner, which supplies the data. For a WRITE request, if any other L1 caches hold a copy of the cache line, coherence messages are sent to each of them requesting that they invalidate their copies. The requesting L1 cache acquires the block in exclusive state only after all invalidation messages have been acknowledged.

  Hop imbalance is quite common in a directory-based protocol. To exploit this imbalance, the critical messages can be sent on fast wires to increase performance and send noncritical messages on slow wires to save power. For the sake of this discussion, it is assumed that the hop latencies of different wires are in the following ratio: L-wire : B-wire : PW-wire :: 1 : 2 : 3

Proposal I: *Read exclusive request for block in shared state*

In this case, the L2 cache's copy is clean, so it provides the data to the requesting L1 and invalidates all shared copies. When the requesting L1 receives the reply message from the L2, it collects invalidation acknowledgment messages from the other L1s before returning the data to the processor core[1]. Figure 5.1 depicts all generated messages.

The reply message from the L2 requires only one hop, while the invalidation process requires two hops – an example of hop imbalance. Since there is no benefit to receiving the cache line early, latencies for each hop can be chosen so as to equalize communication latency for the cache line and the acknowledgment messages. Acknowledgment messages include identifiers so they can be matched against the outstanding request in the L1's MSHR. Since there are only a few outstanding requests in the system, the identifier requires few bits, allowing the acknowledgment to be transferred on a few low-latency L-Wires. Simultaneously, the data block transmission from the L2 can happen on low-power PW-Wires and still finish before the arrival of the acknowledgments. This strategy improves per-

[1]Some coherence protocols may not impose all of these constraints, thereby deviating from a sequentially consistent memory model.

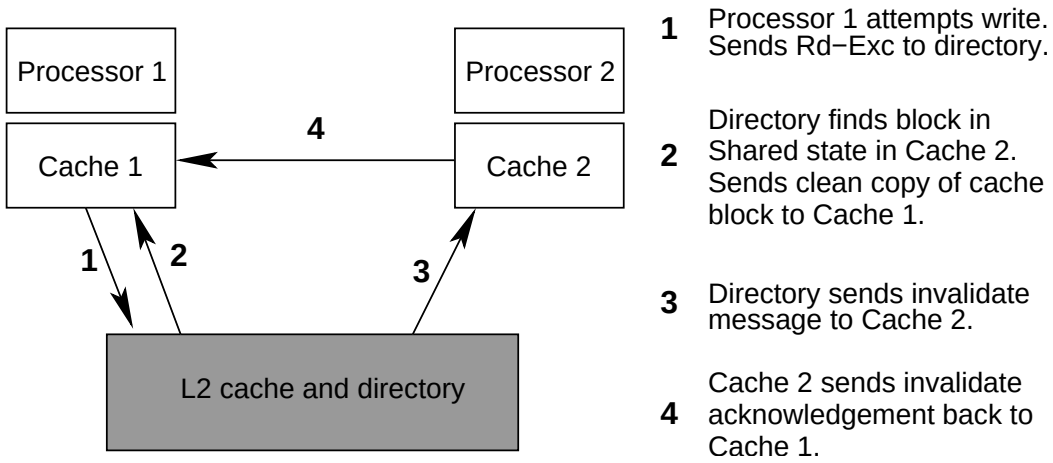


Figure 5.1. Read exclusive request for a shared block in MESI protocol

formance (because acknowledgments are often on the critical path) and reduces power consumption (because the data block is now transferred on power-efficient wires). While circuit designers have frequently employed different types of wires within a circuit to reduce power dissipation without extending the critical path, the proposals in this chapter represent some of the first attempts to exploit wire properties at the architectural level.

Proposal II: *Read request for block in exclusive state*

In this case, the value in the L2 is likely to be stale and the following protocol actions are taken. The L2 cache sends a speculative data reply to the requesting L1 and forwards the read request as an intervention message to the exclusive owner. If the cache copy in the exclusive owner is clean, an acknowledgment message is sent to the requesting L1, indicating that the speculative data reply from the L2 is valid. If the cache copy is dirty, a response message with the latest data is sent to the requesting L1 and a write-back message is sent to the L2. Since the requesting L1 cannot proceed until it receives a message from the exclusive owner, the speculative data reply from the L2 (a single hop transfer) can be sent on slower PW-Wires. The forwarded request to the exclusive owner is on the critical path, but includes the block address. It is therefore not eligible for transfer on low-bandwidth L-Wires. If the owner's copy is in the exclusive clean state, a short acknowledgment message to the requester can be sent on L-Wires. If the owner's copy is dirty, the cache block can be sent over B-Wires, while the low priority writeback to the L2 can happen on PW-Wires. With the above mapping, the critical path is accelerated by using faster L-Wires, while also lowering power consumption by sending noncritical data on PW-Wires. The above protocol actions apply even in the case when a read-exclusive request is made for a block in the exclusive state.

Proposal III: *NACK messages*

When the directory state is busy, incoming requests are often NACKed by the home directory, i.e., a negative acknowledgment is sent to the requester rather than buffering the request. Typically, the requesting cache controller re-issues the

request and the request is serialized in the order in which it is actually accepted by the directory. A NACK message can be matched by comparing the request id (MSHR index) rather than the full address, so a NACK is eligible for transfer on low-bandwidth L-Wires. If load at the home directory is low, it will likely be able to serve the request when it arrives again, in which case, sending the NACK on fast L-Wires can improve performance. In contrast, when load is high, frequent backoff-and-retry cycles are experienced. In this case, fast NACKs only increase traffic levels without providing any performance benefit. In this case, in order to save power, NACKs can be sent on PW-Wires.

Proposal IV: *Unblock and write control messages*

Some protocols [83] employ unblock and write control messages to reduce implementation complexity. For every read transaction, a processor first sends a request message that changes the L2 cache state into a transient state. After receiving the data reply, it sends an unblock message to change the L2 cache state back to a stable state. Similarly, write control messages are used to implement a 3-phase writeback transaction. A processor first sends a control message to the directory to order the writeback message with other request messages. After receiving the writeback response from the directory, the processor sends the data. This avoids a race condition in which the processor sends the writeback data while a request is being forwarded to it. Sending unblock messages on L-Wires can improve performance by reducing the time cache lines are in busy states. Write control messages (writeback request and writeback grant) are not on the critical path, although they are also eligible for transfer on L-Wires. The choice of sending writeback control messages on L-Wires or PW-Wires represents a power-performance trade-off.

- **Write-Invalidate Bus-Based Protocol**

Next,the techniques that apply to bus-based snooping protocols are examined.

Proposal V: *Signal wires*

In a bus-based system, three wired-OR signals are typically employed to avoid

involving the lower/slower memory hierarchy [39]. Two of these signals are responsible for reporting the state of snoop results and the third indicates that the snoop result is valid. The first signal is asserted when any L1 cache, besides the requester, has a copy of the block. The second signal is asserted if any cache has the block in exclusive state. The third signal is an inhibit signal, asserted until all caches have completed their snoop operations. When the third signal is asserted, the requesting L1 and the L2 can safely examine the other two signals. Since all of these signals are on the critical path, implementing them using low-latency L-Wires can improve performance.

Proposal VI: *Voting wires*

Another design choice is whether to use cache-to-cache transfers if the data is in the shared state in a cache. The Silicon Graphics Challenge [49] and the Sun Enterprise use cache-to-cache transfers only for data in the modified state, in which case there is a single supplier. On the other hand, in the full Illinois MESI protocol, a block can be preferentially retrieved from another cache rather than from memory. However, when multiple caches share a copy, a "voting" mechanism is required to decide which cache will supply the data, and this voting mechanism can benefit from the use of low latency wires.

5.2.2 Protocol-independent Techniques

Proposal VII: *Narrow Bit-Width Operands for Synchronization Variables*

Synchronization is one of the most important factors in the performance of a parallel application. Synchronization is not only often on the critical path, but it also contributes a large percentage (up to 40%) of coherence misses [74]. Locks and barriers are the two most widely used synchronization constructs. Both of them use small integers to implement mutual exclusion. Locks often toggle the synchronization variable between zero and one, while barriers often linearly increase a barrier variable from zero to the number of processors taking part in the barrier operation. Such data transfers have limited bandwidth needs and can benefit from using L-Wires.

This optimization can be further extended by examining the general problem of cache line compaction. For example, if a cache line is comprised mostly of 0 bits, trivial data compaction algorithms may reduce the bandwidth needs of the cache line, allowing it to be transferred on L-Wires instead of B-Wires. If the wire latency difference between the two wire implementations is greater than the delay of the compaction/de-compaction algorithm, performance improvements are possible.

Proposal VIII: *Assigning Writeback Data to PW-Wires*
Writeback data transfers result from cache replacements or external request/intervention messages. Since writeback messages are rarely on the critical path, assigning them to PW-Wires can save power without incurring significant performance penalties.

Proposal IX: *Assigning Narrow Messages to L-Wires*
Coherence messages that include the data block address or the data block itself are many bytes wide. However, many other messages, such as acknowledgments and NACKs, do not include the address or data block and only contain control information (source/destination, message type, MSHR id, etc.). Such narrow messages can be always assigned to low latency L-Wires to accelerate the critical path.

5.3 Implementation Complexity

5.3.1 Overhead in Heterogeneous Interconnect Implementation

In a conventional multiprocessor interconnect, a subset of wires are employed for addresses, a subset for data, and a subset for control signals. Every bit of communication is mapped to a unique wire. When employing a heterogeneous interconnect, a communication bit can map to multiple wires. For example, data returned by the L2 in response to a read-exclusive request may map to B-Wires or PW-Wires depending on whether there are other sharers for that block (Proposal I). Thus, every wire must be associated with a multiplexer and de-multiplexer.

The entire network operates at the same fixed clock frequency, which means that the number of latches within every link is a function of the link latency. Therefore, PW-Wires have to employ additional latches, relative to the baseline B-Wires.

Dynamic power per latch at 5GHz and 65nm technology is calculated to be 0.1mW, while leakage power per latch equals 19.8μW [69]. The power per unit length for each wire is computed in the next section. Power overheads due to these latches for different wires are tabulated in Table 5.1. Latches impose a 2% overhead within B-Wires, but a 13% overhead within PW-Wires.

The proposed model also introduces additional complexity in the routing logic. The base case router employs a cross-bar switch and 8-entry message buffers at each input port. Whenever a message arrives, it is stored in the input buffer and routed to an allocator that locates the output port and transfers the message. In case of a heterogeneous model, three different buffers are required at each port to store L, B, and PW messages separately. In this simulation, three 4-entry message buffers for each port is employed. The size of each buffer is proportional to the flit size of the corresponding set of wires. For example, a set of 24 L-Wires employs a 4-entry message buffer with a word size of 24 bits. The power calculations also include the fixed additional overhead associated with these small buffers as opposed to a single larger buffer employed in the base case. In this proposed processor model, the dynamic characterization of messages happens only in the processors and intermediate network routers cannot re-assign a message to a different set of wires. While this may have a negative effect on performance in a highly utilized network, it is chosen to keep the routers simple and not implement such a feature. For a network employing virtual channel flow control, each set of wires in the heterogeneous network link is treated as a separate physical channel and the same

Table 5.1. Power characteristics of different wire implementations. For calculating the power/length, activity factor α (described in Table 5.3) is assumed to be 0.15. The above latch spacing values are for a 5GHz network.

Wire Type	Power/Length mW/mm	Latch Power mW/latch	Latch Spacing mm	Total Power/10mm mW/10mm
B-Wire – 8X plane	1.4221	0.119	5.15	14.46
B-Wire – 4X plane	1.5928	0.119	3.4	16.29
L-Wire – 8X plane	0.7860	0.119	9.8	7.80
PW-wire – 4X plane	0.4778	0.119	1.7	5.48

number of virtual channels are maintained per physical channel. Therefore, the heterogeneous network has a larger total number of virtual channels and the routers require more state fields to keep track of these additional virtual channels. To summarize, the additional overhead introduced by the heterogeneous model comes in the form of potentially more latches and greater routing complexity.

5.3.2 Overhead in Decision Process

The decision process in selecting the right set of wires is minimal. For example, in Proposal I, an OR function on the directory state for that block is enough to select either B- or PW-Wires. In Proposal II, the decision process involves a check to determine if the block is in the exclusive state. To support Proposal III, we need a mechanism that tracks the level of congestion in the network (for example, the number of buffered outstanding messages). There is no decision process involved for Proposals IV, V, VI and VIII. Proposals VII and IX require logic to compute the width of an operand, similar to logic used in the PowerPC 603 [50] to determine the latency for integer multiply.

5.3.3 Overhead in Cache Coherence Protocols

Most coherence protocols are already designed to be robust in the face of variable delays for different messages. For protocols relying on message order within a virtual channel, each virtual channel can be made to consist of a set of L-, B-, and PW-message buffers. A multiplexer can be used to activate only one type of message buffer at a time to ensure correctness. For other protocols that are designed to handle message re-ordering within a virtual channel, it is proposed to employ one dedicated virtual channel for each set of wires to fully exploit the benefits of a heterogeneous interconnect. In all proposed innovations, a data packet is not distributed across different sets of wires. Therefore, different components of an entity do not arrive at different periods of time, thereby eliminating any timing problems. It may be worth considering sending the critical word of a cache line on L-Wires and the rest of the cache line on PW-Wires. Such a proposal may entail

nontrivial complexity to handle corner cases and is not discussed further in this work.

In a snooping bus-based coherence protocol, transactions are serialized by the order in which addresses appear on the bus. None of these proposed innovations for snooping protocols affect the transmission of address bits (address bits are always transmitted on B-Wires), so the transaction serialization model is preserved.

5.4 Methodology

5.4.1 Simulator

A 16-core CMP with the Virtutech Simics full-system functional execution-driven simulator [80] and a timing infrastructure GEMS [82] is simulated. GEMS can simulate both in-order and out-of-order processors. In most studies, the in-order blocking processor model provided by Simics to drive the detailed memory model (Ruby) for fast simulation is used. Ruby implements a one-level MOESI directory cache coherence protocol with migratory sharing optimization [38, 110].

All processor cores share a noninclusive L2 cache, which is organized as a nonuniform cache architecture (NUCA) [58]. Ruby can also be driven by an out-of-order processor module called Opal, and the impact of the processor cores on the heterogeneous interconnect in Section 5.6.1 is reported. Opal is a timing-first simulator that implements the performance sensitive aspects of an out of order processor but ultimately relies on Simics to provide functional correctness. The Opal is configured to model the processor described in Table 5.2 and use an aggressive implementation of sequential consistency.

To test the ideas, a workload consisting of all programs from the SPLASH-2 [127] benchmark suite is employed. The programs were run to completion, but all experimental results reported in this paper are for the parallel phases of these applications. A default input set is used for most programs except fft and radix. Since the default working sets of these two programs are too small, the working set of fft is increased to 1M data points and that of radix to 4M keys.

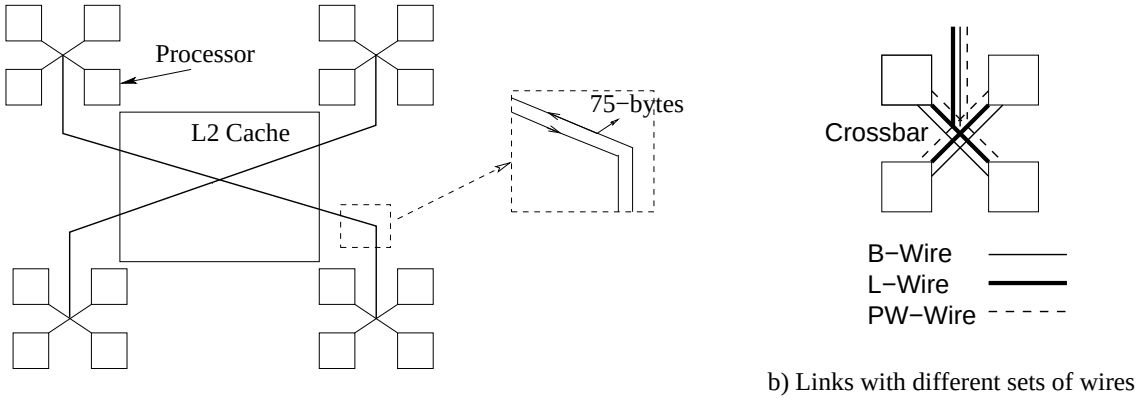
Table 5.2. System configuration.

Parameter	Value
number of cores	16
clock frequency	5GHz
pipeline width	4-wide fetch and issue
pipeline stages	11
cache block size	64 Bytes
split L1 I & D cache	128KB, 4-way
shared L2 cache	8MBytes, 4-way, 16-banks non-inclusive NUCA
memory/dir controllers	30 cycles
interconnect link latency	4 cycles (one-way) for the baseline 8X-B-Wires
DRAM latency	400 cycles
memory bank capacity	1 GByte per bank
latency to mem controller	100 cycles

5.4.2 Interconnect Configuration

This section describes details of the interconnect architecture and the methodology employed for calculating the area, delay, and power values of the interconnect. All the power and delay calculations are based on 65nm process technology with 10 metal layers, 4 layers in 1X plane and 2 layers, in each 2X, 4X, and 8X plane [69]. For this study a crossbar based hierarchical interconnect structure is employed to connect the cores and L2 cache (Figure 5.2(a)), similar to that in SGI's NUMALink-4 [1]. The effect of other interconnect topologies is discussed in the sensitivity analysis. In the base case, each link in Figure 5.2(a) consists of (in each direction) 64-bit address wires, 64-byte data wires, and 24-bit control wires. The control signals carry source, destination, signal type, and Miss Status Holding Register (MSHR) id. All wires are fully pipelined. Thus, each link in the interconnect is capable of transferring 75 bytes in each direction. Error Correction Codes (ECC) account for another 13% overhead in addition to the above mentioned wires [89]. All the wires of the base case are routed as B-Wires in the 8X plane.

As shown in Figure 5.2(b), the proposed heterogeneous model employs additional wire types within each link. In addition to B-Wires, each link includes low-latency, low-bandwidth L-Wires and high-bandwidth, high-latency, power-efficient,



a) Hierarchical network topology for 16−core CMP

b) Links with different sets of wires

Figure 5.2. Interconnect model used for coherence transactions in a sixteen-core CMP.

PW-Wires. The number of L- and PW-Wires that can be employed is a function of the available metal area and the needs of the coherence protocol. In order to match the metal area with the baseline, each uni-directional link within the heterogeneous model is designed to be made up of 24 L-Wires, 512 PW-Wires, and 256 B-Wires (the base case has 600 B-Wires, not counting ECC). In a cycle, three messages may be sent, one on each of the three sets of wires. The bandwidth, delay, and power calculations for these wires are discussed subsequently.

Table 5.3 summarizes the different types of wires and their area, delay, and power characteristics. The area overhead of the interconnect can be mainly attributed to repeaters and wires. The wire width and spacing (based on ITRS projections)are used to calculate the effective area for minimum-width wires in the 4X and 8X plane. L-Wires are designed to occupy four times the area of minimum-width 8X-B-Wires.

Delay: The wire model is based on the RC models proposed in [14, 54, 87]. The delay per unit length of a wire with optimally placed repeaters is given by equation (5.1), where R_{wire} is resistance per unit length of the wire, C_{wire} is capacitance per unit length of the wire, and $FO1$ is the fan-out of one delay:

$$Latency_{wire} = 2.13\sqrt{R_{wire}C_{wire}FO1} \tag{5.1}$$

Table 5.3. Area, delay, and power characteristics of different wire implementations.

Wire Type	Relative Latency	Relative Area ($wireWidth + spacing$)	Dynamic Power (W/m) α = Switching Factor	Static Power W/m
B-Wire (8X plane)	$1x$	$1x$	2.65α	1.0246
B-Wire (4X plane)	$1.6x$	$0.5x$	2.9α	1.1578
L-Wire (8X plane)	$0.5x$	$4x$	1.46α	0.5670
PW-Wire (4X plane)	$3.2x$	$0.5x$	0.87α	0.3074

R_{wire} is inversely proportional to wire width, while C_{wire} depends on the following three components: (i) fringing capacitance that accounts for the capacitance between the side wall of the wire and substrate, (ii) parallel plate capacitance between the top and bottom layers of the metal that is directly proportional to the width of the metal, (iii) parallel plate capacitance between the adjacent metal wires that is inversely proportional to the spacing between the wires. The C_{wire} value for the top most metal layer at 65nm technology is given by equation (5.2) [87].

$$C_{wire} = 0.065 + 0.057W + 0.015/S(fF/\mu) \tag{5.2}$$

The relative delays for different types of wires are derived by tuning width and spacing in the above equations. A variety of width and spacing values can allow L-Wires to yield a two-fold latency improvement at a four-fold area cost, relative to 8X-B-Wires. In order to reduce power consumption, a wire implementation is selected where the L-Wire's width was twice that of the minimum width and the spacing was six times as much as the minimum spacing for the 8X metal plane.

Power: The total power consumed by a wire is the sum of three components (dynamic, leakage, and short-circuit power). Equations derived by Banerjee and Mehrotra [14] are used to derive the power consumed by L- and B-Wires. These equations take into account optimal repeater size/spacing and wire width/spacing. PW-Wires are designed to have twice the delay of 4X-B-Wires. At 65nm technology, for a delay penalty of 100%, smaller and widely-spaced repeaters enable power reduction by 70% [14].

Routers: Crossbars, buffers, and arbiters are the major contributors for router power [120]:

$$E_{router} = E_{buffer} + E_{crossbar} + E_{arbiter} \tag{5.3}$$

The capacitance and energy for each of these components is based on analytical models proposed by Wang et al. [120]. A 5x5 matrix crossbar that employs a tristate buffer connector is modeled. As described in Section 5.3, buffers are modeled for each set of wires with word size corresponding to flit size. Table 5.3 shows the peak energy consumed by each component of the router for a single 32-byte transaction.

5.5 Results

The simulations are restricted to directory-based protocols. The effect of proposals pertaining is modeled to such a protocol: I, III, IV, VIII, IX. Proposal-II optimizes speculative reply messages in MESI protocols, which are not implemented within GEMS' MOESI protocol. Evaluations involving compaction of cache blocks (Proposal VII) is left as future work.

Figure 5.3 shows the execution time in cycles for SPLASH2 programs. The first bar shows the performance of the baseline organization that has one interconnect layer of 75 bytes, composed entirely of 8X-B-Wires. The second shows the performance of the heterogeneous interconnect model in which each link consists of 24-bit L-wires, 32-byte B-wires, and 64-byte PW-wires. Programs such as LU-Non-continuous, Ocean-Non-continuous, and Raytracing yield significant improvements in performance. These performance numbers can be analyzed with the help of Figure 5.4 that shows the distribution of different transfers that happen on the interconnect. Transfers on L-Wires can have a huge impact on performance, provided they are on the program critical path. LU-Non-continuous, Ocean-Non-continuous, Ocean-Continuous, and Raytracing experience the most transfers on L-Wires. However, the performance improvement of Ocean-Continuous is very low compared to other benchmarks. This can be attributed to the fact that Ocean-Continuous incurs the most L2 cache misses and is mostly memory bound. The transfers on PW-Wires have a negligible effect on performance for all benchmarks. This is

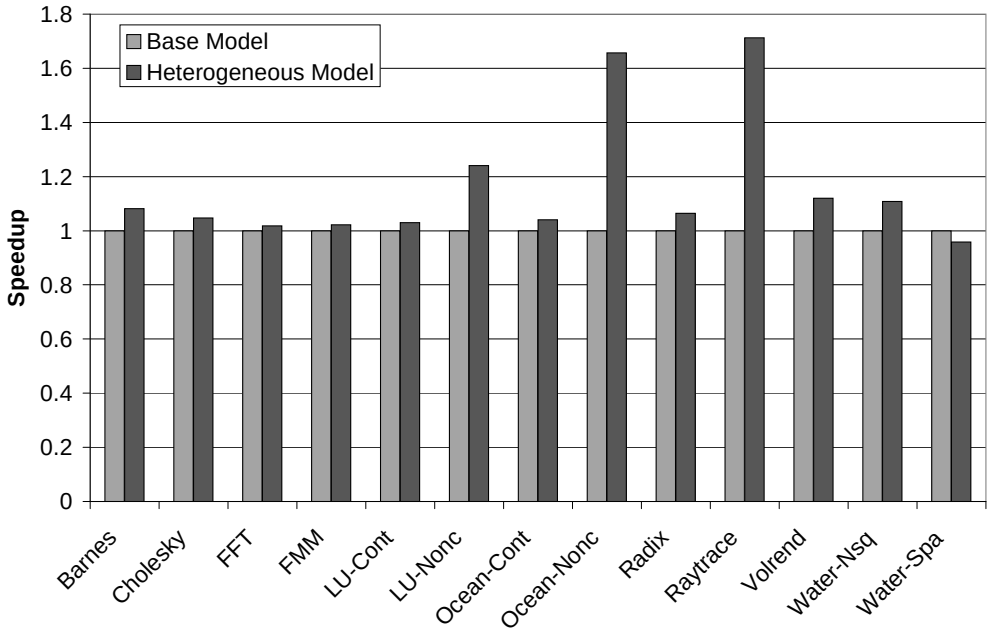


Figure 5.3. Speedup of heterogeneous interconnect

because PW-Wires are employed only for writeback transfers that are always off the critical path. On average, a 11.2% improvement in performance is observed, compared to the baseline, by employing heterogeneity within the network.

Proposals I, III, IV, and IX exploit L-Wires to send small messages within the protocol, and contribute 2.3, 0, 60.3, and 37.4 per cent, respectively, to total L-Wire traffic. A per-benchmark breakdown is shown in Figure 5.5. Proposal-I optimizes the case of a read exclusive request for a block in shared state, which is not very common in the SPLASH2 benchmarks. It is expected that the impact of Proposal-I will be much higher in commercial workloads where cache-to-cache misses dominate. Proposal-III and Proposal-IV impact NACK, unblocking, and writecontrol messages. These messages are widely used to reduce the implementation complexity of coherence protocols. In GEMS' MOESI protocol, NACK messages are only used to handle the race condition between two write-back messages, which are negligible in this study (causing the zero contribution of Proposal-III). Instead, the protocol implementation relies heavily on unblocking and writecontrol messages to maintain the order between read and write transactions, as discussed in Section 5.2.1. The

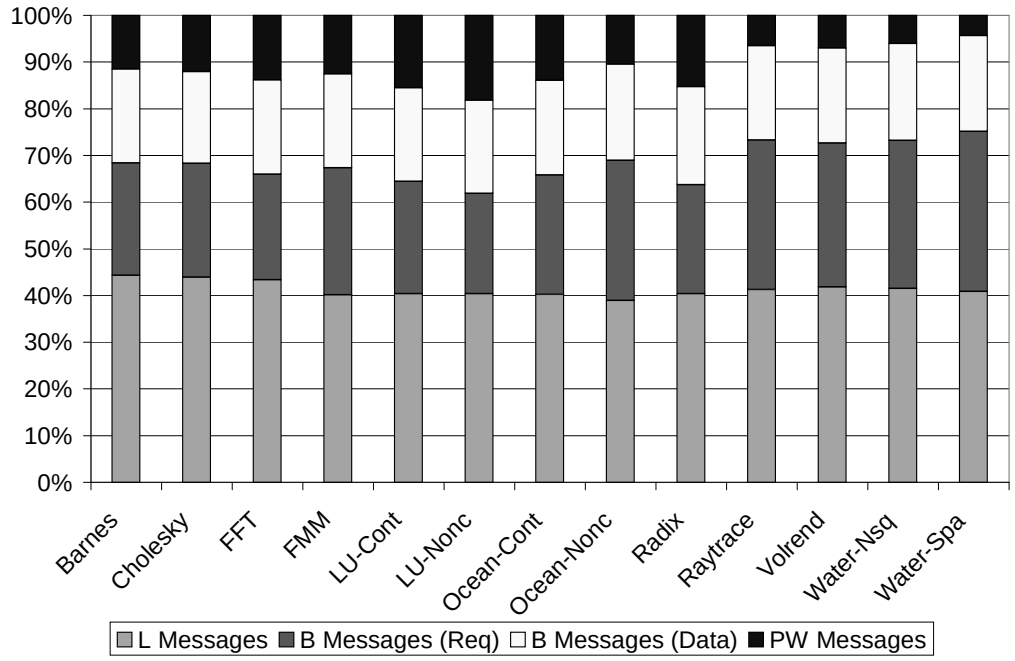


Figure 5.4. Distribution of messages on the heterogeneous network. B-Wire transfers are classified as Request and Data.

frequency of occurrence of NACK, unblocking, and writecontrol messages depends on the protocol implementation, but the sum of these messages are expected to be relatively constant in different protocols and play an important role in L-wire optimizations. Proposal-IX includes all other acknowledgment messages eligible for transfer on L-Wires.

It is observed that the combination of proposals I, III, IV, and IX caused a performance improvement more than the sum of improvements from each individual proposal. A parallel benchmark can be divided into a number of phases by synchronization variables (barriers), and the execution time of each phase can be defined as the longest time any thread spends from one barrier to the next. Optimizations applied to a single thread may have no effect if there are other threads on the critical path. However, a different optimization may apply to the threads on the critical path, reduce their execution time, and expose the performance of other threads and the optimizations that apply to them. Since different threads take different data paths, most parallel applications show nontrivial workload imbalance [76].

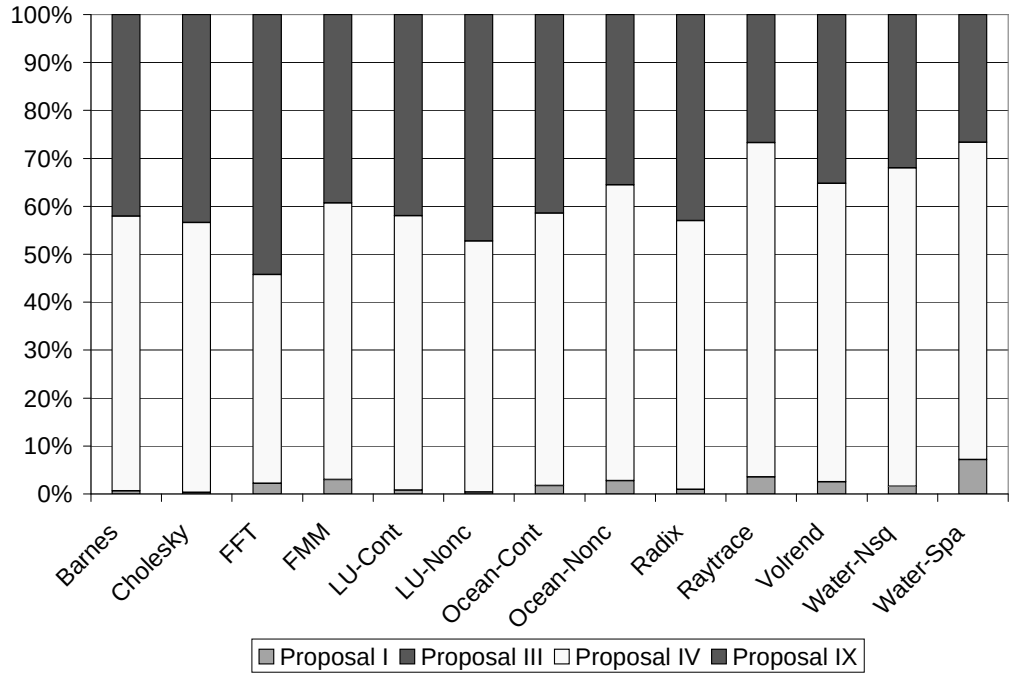


Figure 5.5. Distribution of L-message transfers across different proposals.

Therefore, employing one proposal might not speedup all threads on the critical path, but employing all applicable proposals can probably optimize threads on every path, thereby reducing the total barrier to barrier time.

Figure 5.6 shows the improvement in network energy due to the heterogeneous interconnect model. The first bar shows the reduction in network energy and the second bar shows the improvement in the overall processor $Energy \times Delay^2$ (ED^2) metric. Other metrics in the $E - D$ space can also be computed with data in Figures 5.6 and 5.3. To calculate ED^2, it is assumed that the total power consumption of the chip is 200W, of which the network power accounts for 60W. The energy improvement in the heterogeneous case comes from both L and PW transfers. Many control messages that are sent on B-Wires in the base case are sent on L-Wires in the heterogeneous case. As per Table 5.3, the energy consumed by an L-Wire is less than the energy consumed by a B-Wire. However, due to the small sizes of these messages, the contribution of L-messages to the total energy

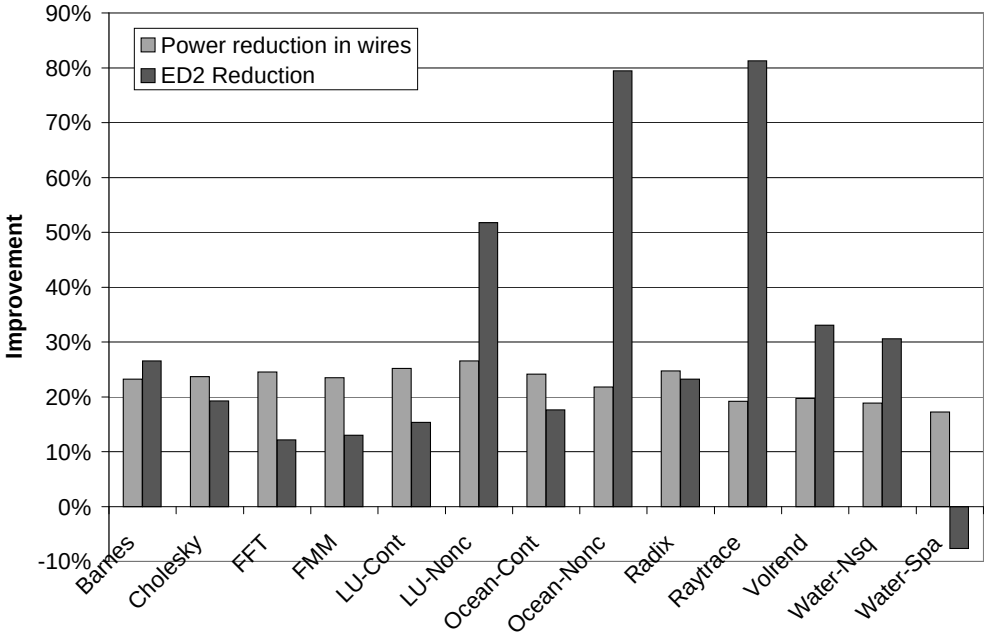


Figure 5.6. Improvement in link energy and ED^2.

savings is negligible. Overall, the heterogeneous network results in a 22% saving in network energy and a 30% improvement in ED^2.

5.6 Sensitivity Analysis

In this subsection, the impact of processor cores, link bandwidth, routing algorithm, and network topology on the heterogeneous interconnect are discussed.

5.6.1 Out-of-order/In-order Processors

To test the ideas with an out-of-order processor, the Opal is configured to model the processor described in Table 5.2 and only reports the results of the first 100M instructions in the parallel sections[2].

Figure 5.7 shows the performance speedup of the heterogeneous interconnect over the baseline. All benchmarks except Ocean-Noncontinuous demonstrate dif-

[2]Simulating the entire program takes nearly a week and there exist no effective toolkits to find the representative phases for parallel benchmarks. LU-Noncontinuous and Radix were not compatible with the Opal timing module.

ferent degrees of performance improvement, which leads to an average speedup of 9.3%. The average performance improvement is less than what is observed in a system employing in-order cores (11.2%). This can be attributed to the greater tolerance that an out-of-order processor has to long instruction latencies.

5.6.2 Link Bandwidth

The heterogeneous network poses more constraints on the type of messages that can be issued by a processor in a cycle. It is therefore likely to not perform very well in a bandwidth-constrained system. To verify this, a base case is modeled where every link has only 80 8X-B-Wires and a heterogeneous case where every link is composed of 24 L-Wires, 24 8X-B-Wires, and 48 PW-Wires (almost twice the metal area of the new base case). Benchmarks with higher network utilizations suffered significant performance losses. In these experiments, raytracing has the maximum messages/cycle ratio and the heterogeneous case suffered a 27% performance loss, compared to the base case (in spite of having twice the metal area). The heterogeneous interconnect performance improvement for Ocean Non-continuous and LU Non-continuous is 12% and 11%, as against 39% and 20% in the high-bandwidth simulations. Overall, the heterogeneous model performed 1.5% worse than the base case.

5.6.3 Routing Algorithm

The simulations thus far have employed adaptive routing within the network. Adaptive routing alleviates the contention problem by dynamically routing messages based on the network traffic. It is found that deterministic routing degraded performance by about 3% for most programs for systems with the baseline and with the heterogeneous network. Raytracing is the only benchmark that incurs a significant performance penalty of 27% for both networks.

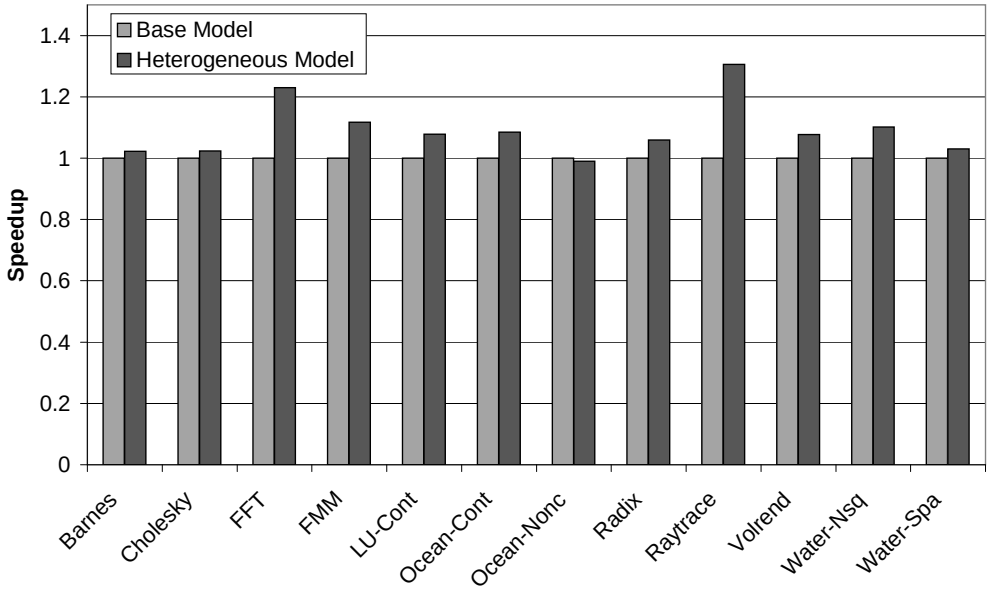


Figure 5.7. Speedup of heterogeneous interconnect when driven by OoO cores (Opal and Ruby)

5.6.4 Network Topology

The default interconnect thus far was a two-level tree based on SGI's NUMALink-4 [1]. To test the sensitivity of these results to the network topology, a 2D-torus interconnect resembling that in the Alpha 21364 [17] is also examined. As shown in Figure 5.8, each router connects to 4 links that connect to 4 neighbors in the torus, and wraparound links are employed to connect routers on the boundary.

The proposed mechanisms show much less performance benefit (1.3% on average) in the 2D torus interconnect than in the two-level tree interconnect as shown in Figure 5.9. The main reason is that the decision process in selecting the right set of wires calculates hop imbalance at the coherence protocol level without considering the physical hops a message takes on the mapped topology. For example, in a 3-hop transaction as shown in Figure 5.1, the one-hop message may take 4 physical hops while the 2-hop message may also take 4 physical hops. In this case, sending the 2-hop message on the L-Wires and the one-hop message on the PW-Wires will actually lower performance.

This is not a first-order effect in the two-level tree interconnect, where most hops take 4 physical hops. However, the average distance between two processors

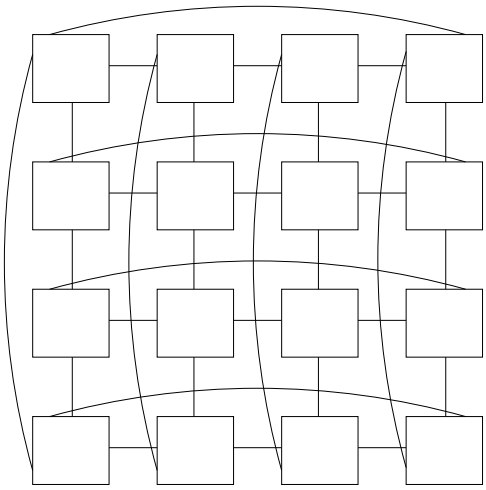


Figure 5.8. 2D Torus topology

in the 2D torus interconnect is 2.13 physical hops with a standard deviation of 0.92 hops. In an interconnect with such high standard deviation, calculating hop imbalance based on protocol hops is inaccurate. For future work, it is planed to develop a more accurate decision process that considers source id, destination id, and interconnect topology to dynamically compute an optimal mapping to wires.

5.7 Related Work

To the best of my knowledge, only two other bodies of work have attempted to exploit different types of interconnects at the microarchitecture level. Beckmann and Wood [20, 21] propose speeding up access to large L2 caches by introducing transmission lines between the cache controller and individual banks. Nelson et al. [95] propose using optical interconnects to reduce inter-cluster latencies in a clustered architecture where clusters are widely-spaced in an effort to alleviate power density. This is the first work that exploits heterogeneous full-swing wires in a multicore environment to reduce coherence overhead.

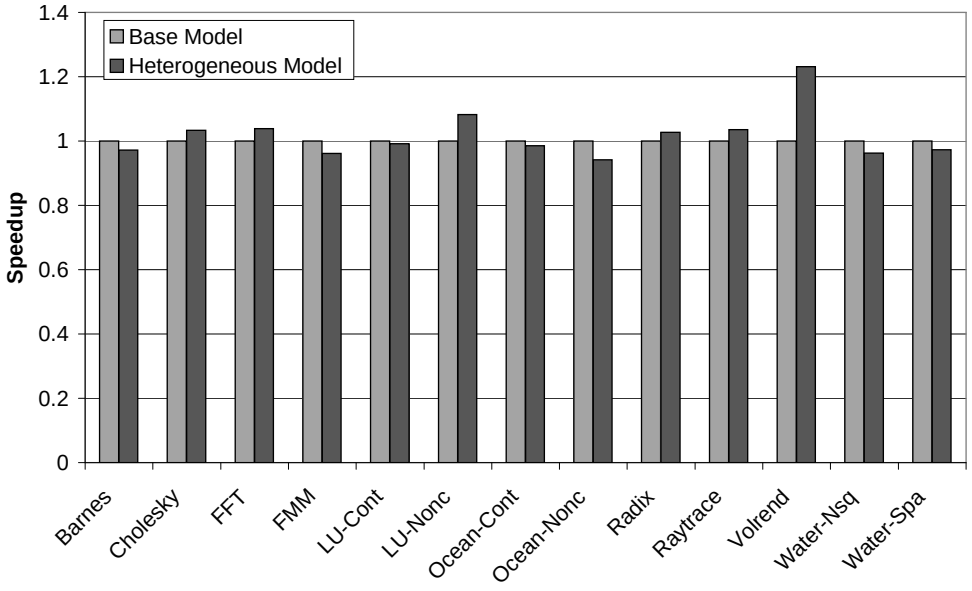


Figure 5.9. Heterogeneous interconnect speedup

5.8 Summary

Coherence traffic in a chip multiprocessor has diverse needs. Some messages can tolerate long latencies, while others are on the program critical path. Further, messages have varied bandwidth demands. The specific needs of these messages can be met by embracing a heterogeneous network. This chapter presents numerous novel techniques that can exploit a heterogeneous interconnect to simultaneously improve performance and reduce power consumption.

The evaluation of the proposed techniques targeted at a directory-based protocol shows that a large fraction of messages have low bandwidth needs and can be transmitted on low latency wires, thereby yielding a performance improvement of 11.2%. At the same time, a 22.5% reduction in interconnect energy is observed by transmitting noncritical data on power-efficient wires. The complexity cost is marginal as the mapping of messages to wires entails simple logic.

There may be several other applications of heterogeneous interconnects within a CMP. For example, in the *Dynamic Self Invalidation* scheme proposed by Lebeck et al. [73], the self-invalidate [71, 73] messages can be effected through power-efficient PW-Wires. In a processor model implementing token coherence, the low-bandwidth

token messages [83] are often on the critical path and, thus, can be effected on L-Wires. A recent study by Huh et al. [56] reduces the frequency of false sharing by employing incoherent data. For cache lines suffering from false sharing, only the sharing states need to be propagated and such messages are a good match for low-bandwidth L-Wires. Overall, a heterogeneous interconnect, in addition to helping to build an efficient cache hierarchy, also alleviates the overhead of the inter-core communications happening through the cache hierarchy.

CHAPTER 6

WIRE MANAGEMENT IN A CLUSTERED ARCHITECTURE

In the last few chapters, the role of interconnects in designing large lower level on-chip caches is studied. This chapter focuses on application of heterogeneous network to accelerate L1 accesses. Since the access time of L1 is potentially wire-limited in a high-ILP clustered architecture, the rest of the chapter assumes a clustered microarchitecture for the out-of-order core. In a clustered architecture, on-chip network also carries register values, thus lending itself to a few additional optimizations that are not strictly tied to L1 cache.

6.1 Clustered Architecture

One of the biggest challenges for computer architects is the design of billion-transistor architectures that yield high parallelism, high clock speeds, low design complexity, and low power. There appears to be a consensus among several research groups [4, 12, 16, 28, 36, 47, 54, 61, 62, 63, 66, 93, 96, 106, 114] that a *partitioned architecture* is the best approach to achieving these design goals.

Partitioned architectures consist of many small and fast computational units connected by a communication fabric. A computational unit is commonly referred to as a *cluster* and is typically comprised of a limited number of ALUs, local register storage, and a buffer for instruction issue. Since a cluster has limited resources and functionality, it enables fast clocks, low power, and low design effort. Abundant transistor budgets allow the incorporation of many clusters on a chip. The instructions of a single program are distributed across the clusters, thereby enabling high parallelism. Since it is impossible to localize all dependent instructions to a single cluster, data is frequently communicated between clusters over the inter-cluster

communication fabric. Depending on the workloads, different flavors of partitioned architectures can exploit instruction-level, data-level, and thread-level parallelism (ILP, DLP, and TLP).

One of the biggest bottlenecks in a clustered architecture is the inter-core communication overhead. To alleviate the high performance penalty of long wire delays at future technologies, most research efforts have concentrated on reducing the number of communications through intelligent instruction and data assignment to clusters. Such an assignment can be accomplished either at compile-time [47, 61, 66, 93, 96, 106, 114] or at run-time [4, 12, 16, 28, 36]. However, in spite of our best efforts, global communication is here to stay. For a dynamically scheduled 4-cluster system (described in Sections 6.3 and 6.4), performance degrades by 12% when the inter-cluster latency is doubled. The papers listed above also report similar slowdowns for high-latency interconnects. Thus, irrespective of the implementation, partitioned architectures experience a large number of global data transfers and performance can be severely degraded if the interconnects are not optimized for low delay.

Since inter-cluster communications happen on long wires with high capacitances, they are responsible for a significant fraction of on-chip power dissipation. Interconnect power is a major problem not only in today's industrial designs, but also in high-performance research prototypes. A recent evaluation by Wang et al. [122] demonstrates that the inter-tile network accounts for 36% of the total energy dissipated in the Raw processor [114]. Hence, by focusing on techniques that reduce interconnect overhead, significant improvement in performance and power savings are possible. To reduce wire delay, a low-latency, low-bandwidth interconnect and design a cache pipeline that employs a subset of the address bits to prefetch data is leveraged out of cache banks. The advantage of the fact is also taken that a number of data transfers involve narrow bit-width operands that can benefit from a low-bandwidth interconnect. Further, improved performance is seen by diverting bursts of interconnect traffic to high-bandwidth high-latency interconnects. These

high-bandwidth interconnects can also be designed to be energy-efficient, enabling significant energy savings in addition to performance improvements.

In this section, the partitioned architecture model that serves as an evaluation platform for this study and the proposed innovations that can take advantage of a heterogeneous interconnect is described.

6.2 The Baseline Partitioned Architecture

Instruction assignment to clusters in a partitioned architecture may happen at compile-time [18, 47, 61, 66, 93, 96, 106], or at run-time [4, 12, 16, 28, 36]. There are advantages to either approach – static techniques entail lower hardware overheads and have access to more information on program dataflow, while dynamic techniques are more reactive to events such as branch mispredicts, cache misses, network congestion, etc. These evaluations employ a dynamically scheduled partitioned architecture. It is expected that these proposals can be applied even to statically scheduled architectures.

This partitioned architecture model dispatches a large window of in-flight instructions from a single-threaded application. A centralized cache implementation is adopted because earlier studies have shown that a centralized cache offers nearly as much performance as a distributed cache while enabling low implementation complexity [11, 52, 101]. The assignment of instructions to clusters happens through a state-of-the-art dynamic instruction steering heuristic [12, 28, 118] that takes the following information into account: data dependences, cluster load imbalance, criticality of operands, and proximity to the data cache. While dispatching an instruction, the steering algorithm assigns weights to each cluster to determine the cluster that is most likely to minimize communication and issue-related stalls. Weights are assigned to a cluster if it produces input operands for the instruction and if it has many empty issue queue entries. Additional weights are assigned to a cluster if it is the producer of the input operand that is predicted to be on the critical path for the instruction's execution. For loads, more weights are assigned to clusters that are closest to the data cache. The steering algorithm assigns the

instruction to the cluster that has the most weights. If that cluster has no free register and issue queue resources, the instruction is assigned to the nearest cluster with available resources.

Results produced within a cluster are bypassed to consumers in that cluster in the same cycle, while communicating the result to consumers in other clusters takes additional cycles. In order to effect the transfer of data between clusters, the instruction decode and rename stage inserts a "copy instruction" [28] in the producing cluster that places the value on the inter-cluster network as soon as the value is made available. Each cluster has a scheduler for the inter-cluster network that is similar in organization to the issue queue and that has an issue bandwidth that matches the maximum number of transfers possible on each link of the network. Similar to the instruction wake-up process in conventional dynamic superscalars, the register tags for the operand are sent on the network ahead of the data so that the dependent instruction can be woken up and can consume the value as soon as it arrives.

For most of the experiments, a processor model is assumed that has four clusters. These four clusters and the centralized data cache are connected through a crossbar network, as shown in Figure 6.1 (a). All links contain a unidirectional interconnect in each direction. The processor model in Figure 6.1 (a) adopts a heterogeneous interconnect where every link in the network is comprised of *B-Wires*, *PW-Wires*, and *L-Wires*. Note that every data transfer has the option to use any one of these sets of wires. Our evaluations show the effects of using interconnects that employ different combinations of these sets of wires. For all processor organizations, the bandwidth requirements to the cache are much higher than bandwidth requirements to the clusters since more than one third of all instructions are loads or stores. Hence, the links going in and out of the cache are assumed to have twice as much area and twice as many wires as the links going in and out of a cluster. If multiple transfers compete for a link in a cycle, one transfer is effected in that cycle, while the others are buffered. Unbounded buffers are assumed at each node of the network. An earlier study [97] has shown that these buffers typically require a modest number

of entries. The aggressive processor models with 16 clusters are also examined. For a 16-cluster system, a hierarchical topology similar to the one proposed by Aggarwal and Franklin [5] is adopted. As shown in Figure 6.1 (b), a set of four clusters is connected through a crossbar, allowing low-latency communication to neighboring clusters. The crossbars are connected with a ring topology. Similar to the 4-cluster system, every link in the network is comprised of wires with different properties.

6.3 Exploiting Heterogeneous Interconnects

6.3.1 Accelerating Cache Access

First, it is examined how low-latency low-bandwidth *L-Wires* can be exploited to improve performance. *L-Wires* are designed by either employing very large wire widths and spacing or by implementing transmission lines. Because of the area overhead associated with L-wires 2, the*L-Wires* is modelled such that 18 *L-Wires* occupy the same metal area as 72 *B-Wires*.

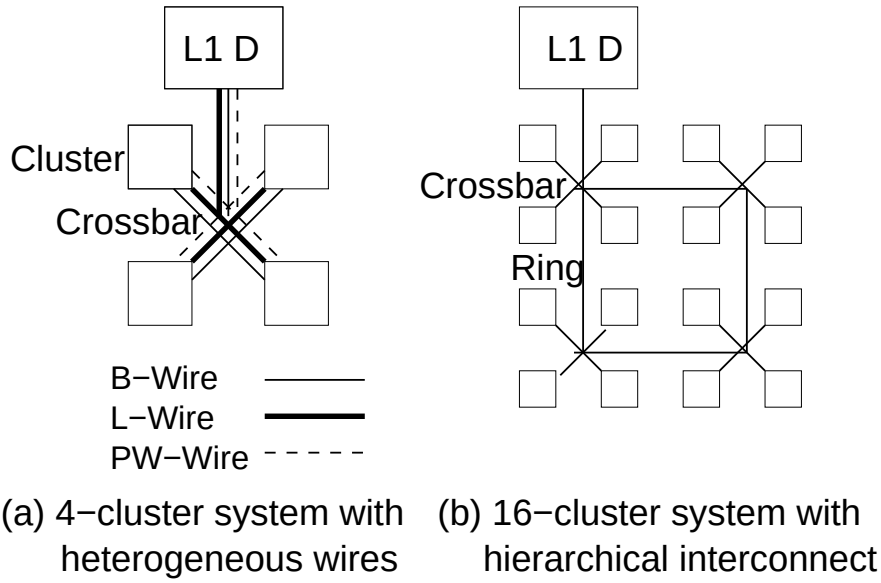


Figure 6.1. Clustered architecture with heterogeneous interconnect. (a) A partitioned architecture model with 4 clusters and a heterogeneous interconnect comprised of *B-*, *L-*, and *PW-Wires*. (b) A 16-cluster system with a hierarchical interconnect. Sets of four clusters are connected with a crossbar and the crossbars are connected in a ring topology.

Consider the behavior of the cache pipeline in the baseline processor. When a cluster executes a load instruction, it computes the effective address and communicates it to the centralized load/store queue (LSQ) and cache. The load/store queue waits until it receives addresses of stores prior to the load in program order, guarantees that there is no memory dependence, and then initiates the cache access. The cost of communication to the cache influences load latency in two ways – (i) it delays the arrival of load addresses at the LSQ, (ii) it delays the arrival of store addresses at the LSQ, thereby delaying the resolution of memory dependences.

To accelerate cache access, the following novel technique is proposed. A subset of the address bits are transmitted on low-latency *L-Wires* to prefetch data out of the L1 cache and hide the high communication cost of transmitting the entire address. After the cluster computes the effective address, the least significant (LS) bits of the address are transmitted on the low-latency *L-Wires*, while the most significant (MS) bits are transmitted on *B-Wires*. The same happens for store addresses. Thus, the LSQ quickly receives the LS bits for loads and stores, while the MS bits take much longer. The early arrival of the partial addresses allows the following optimizations.

The LSQ can effect a partial comparison of load and store addresses with the available LS bits. If the LS bits of the load do not match the LS bits of any earlier store, the load is guaranteed to not have any memory dependence conflicts and it can begin cache access. If the LS bits of the load match the LS bits of an earlier store, it has to wait for the MS bits to arrive before determining if there is a true dependence. A large number of false dependences can also increase contention for the LSQ ports. Fortunately, it is found that false dependences were encountered for fewer than 9% of all loads when employing eight LS bits for the partial address comparison.

To effect an L1 data cache access, the least significant bits of the effective address are used to index into the data and tag RAM arrays and read out a relevant set of cache blocks. The most significant bits of the effective address are used to index into the TLB and the resulting translation is then compared with the tags to select

the appropriate data block and forward it to the cluster. Since the accesses to the cache RAM arrays do not require the most significant bits, the accesses can be initiated as soon as the least significant bits of the address arrive on *L-Wires* (provided the *L-Wires* transmit enough bits to determine the set index).

Similarly, a few bits of the virtual page number can be included in the transfer on the *L-Wires*. This allows TLB access to proceed in parallel with RAM array look-up. The modifications to enable indexing with partial address information are more significant for a CAM structure than a RAM structure. Hence, a highly-associative TLB design may be more amenable to this modified pipeline than a fully-associative one. When the rest of the effective address arrives, tag comparison selects the correct translation from a small subset of candidate translations.

Thus, the transfer of partial address bits on *L-Wires* enables data to be prefetched out of L1 cache and TLB banks and hide the RAM access latency, which is the biggest component in cache access time. If the cache RAM access has completed by the time the entire address arrives, only an additional cycle is spent to detect the correct TLB translation and effect the tag comparison before returning data to the cluster. This overlap of effective address transfer with cache RAM and TLB access can result in a reduction in effective load latency if the latency difference between *L-Wires* and *B-Wires* is significant.

It must be noted that the proposed pipeline works well and yields speedups even if the processor implements some form of memory dependence speculation. The partial address can proceed straight to the L1 cache and prefetch data out of cache banks without going through partial address comparisons in the LSQ if it is predicted to not have memory dependences. To allow cache and TLB index bits to fit in a narrow low-bandwidth interconnect, it might be necessary to make the cache and TLB highly set-associative. For example, 18 *L-Wires* can accommodate 6 bits of tag to identify the instruction in the LSQ, 8 index bits for the L1 data cache, and 4 index bits for the TLB. For the assumed cache and TLB sizes, this corresponds to an associativity of 4 and 8 for the cache and TLB, respectively. If the associativity is reduced, a few more *L-Wires* may be needed.

6.3.2 Narrow Bit-Width Operands

An interconnect composed of *L-Wires* can also be employed for results that can be encoded by a few bits. 18 *L-Wires* can accommodate eight bits of register tag and ten bits of data. A simplest form of data compaction is employed here – integer results between 0 and 1023 are eligible for transfer on *L-Wires*. The hardware required to detect narrow bit-width data can be easily implemented – the PowerPC 603 [50] has hardware to detect the number of leading zeros that is then used to determine the latency for integer multiply. A special case in the transfer of narrow bit-width data is the communication of a branch mispredict back to the front-end. This only involves the branch ID that can be easily accommodated on *L-Wires*, thereby reducing the branch mispredict penalty.

Other forms of data compaction might also be possible, but is not explored here. For example, Yang et al. [129] identify that the eight most frequent values in SPEC95-Int programs account for roughly 50% of all data cache accesses and can be easily encoded by a few bits.

In order to schedule a wake-up operation at the consuming cluster, the register tags are sent before the data itself. For a narrow bit-width operand, the tags have to be sent on *L-Wires*. Hence, the pipeline requires advance knowledge of whether the result can be expressed in 10 bits. For the evaluations, the optimistic assumption made is that this information is available early in the pipeline. A realistic implementation would require inspection of the instruction's input operands or a simple predictor. It is confirmed that a predictor with 8K 2-bit saturating counters, that predicts the occurrence of a narrow bit-width result when the 2-bit counter value is three, is able to identify 95% of all narrow bit-width results. With such a high-confidence predictor, only 2% of all results predicted to be narrow have bit widths greater than 10.

6.3.3 Exploiting PW-Wires

Next, it is examined how *PW-Wires* can be employed to not only reduce contention in other wires, but also reduce energy consumption. The objective here is

to identify those data transfers that can tolerate the higher latency of these wires or to identify situations when the cost of contention on *B-Wires* offsets its wire latency advantage. If a data transfer has the choice of using either *B-Wires* or *PW-Wires*, the following three criteria dictate when a transfer can be effected on the high bandwidth, low energy, high latency *PW-Wires*:

- If the input operands are already ready in a remote register file at the time an instruction is dispatched, the operands are transferred to the instruction's cluster on *PW-Wires*. The rationale here is that there is usually a long gap between instruction dispatch and issue and the long communication latency for the ready input operand can be tolerated.

- Store data is assigned to *PW-Wires*. This can slow the program down only if the store is holding up the commit process or if there is a waiting dependent load. Both are fairly rare cases and a minimal performance impact from adopting this policy is noticed.

- The amount of traffic injected into either interconnect in the past N cycles (N=5 in this simulations) is tracked. If the difference between the traffic in each interconnect exceeds a certain prespecified threshold (10 in this simulations), subsequent data transfers are steered to the less congested interconnect.

Thus, by steering noncritical data towards the high-bandwidth energy-efficient interconnect, little performance degradation is likely to be seen and by steering data away from the congested interconnect, performance improvements can potentially be seen. Most importantly, large savings in interconnect energy can be observed.

6.4 Results

6.4.1 Methodology

The simulator is based on Simplescalar-3.0 [26] for the Alpha AXP ISA. Separate issue queues and physical register files for integer and floating-point streams are

modeled for each cluster. Contention on the interconnects and for memory hierarchy resources (ports, banks, buffers, etc.) are modeled in detail. It is assumed that each cluster has 32 registers (int and fp, each), 15 issue queue entries (int and fp, each), and one functional unit of each kind. While a large ROB size of 480 is used, in-flight instruction windows are typically much smaller as dispatch gets stalled as soon as the processor runs out of physical registers or issue queue entries. The evaluations show results for processor models with four and sixteen clusters. Important simulation parameters are listed in Table 6.1.

23 of the 26 SPEC-2k programs are used with reference inputs as a benchmark set [1]. Each program was simulated for 100 million instructions over simulation windows identified by the Simpoint toolkit [107]. Detailed simulation was carried out for one million instructions to warm up various processor structures before taking measurements.

[1] *Sixtrack*, *Facerec*, and *Perlbmk* were not compatible with the simulation infrastructure.

Table 6.1. Simplescalar simulator parameters.

Fetch queue size	64
Branch predictor	comb. of bimodal and 2-level
Bimodal predictor size	16K
Level 1 predictor	16K entries, history 12
Level 2 predictor	16K entries
BTB size	16K sets, 2-way
Branch mispredict penalty	at least 12 cycles
Fetch width	8 (across up to 2 basic blocks)
Issue queue size	15 per cluster (int and fp, each)
Register file size	32 per cluster (int and fp, each)
Integer ALUs/mult-div	1/1 per cluster
FP ALUs/mult-div	1/1 per cluster
L1 I-cache	32KB 2-way
Memory latency	300 cycles for the first block
L1 D-cache	32KB 4-way set-associative
L2 unified cache	8MB 8-way, 30 cycles
	6 cycles, 4-way word-interleaved
I and D TLB	128 entries, 8KB page size

6.4.2 Latency and Energy Estimates

It is started by assuming that *W-Wires* have the minimum allowed width and spacing for the selected metal layer. Then a *PW-Wire* is designed by reducing the size and number of repeaters. According to the methodology discussed in Chapter 2, roughly 70% of interconnect energy can be saved at 45nm technology while incurring a 20% delay penalty. The *B-Wires* is designed such that each wire has twice as much metal area as a *PW-Wire* and its delay is lower by a factor of 1.5. The delay constraints were able to be met by keeping the width the same as a *W-Wire* and only increasing wire spacing. This strategy also helps us reduce the power consumed in *B-Wires*. Finally, *L-Wires* were designed by increasing the width and spacing of *W-Wires* by a factor of 8. Based on the analysis of Banerjee et al. [14, 87], it is computed that at 45nm technology, $R_L = 0.125R_W$, $C_L = 0.8C_W$, giving us the result that $Delay_L = 0.3Delay_W = 0.25Delay_{PW}$. If *L-Wires* is implemented instead as transmission lines, the improvement in wire delay will be much more. Chang *et al.* [31] report that at 180nm technology, a transmission line is faster than an RC-based repeated wire of the same width by a factor of 4/3. This gap may widen at future technologies. For the purposes of the evaluation, it is restricted to RC-based models, but note that performance and energy improvements can be higher if transmission lines become a cost-effective option.

It is assumed that communication with neighbors through the crossbar takes three cycles for *PW-Wires*. Based on the relative latency estimates above, *B-Wires* and *L-Wires* are 1.5 times and 4 times faster than *PW-Wires*, corresponding to inter-cluster communication latencies of two cycles and one cycle, respectively. When examining a 16-cluster system with a hierarchical interconnect, the latency for a hop on the ring interconnect for *PW-Wires*, *B-Wires*, and *L-Wires* are assumed to be 6, 4, and 2 cycles. The various wire parameters are summarized in Table 6.2. It is assumed that all transfers are fully pipelined. The energy estimates for 45nm are derived from the analysis described by Banerjee et al. [14, 87]. Relative dynamic and leakage energy values are listed in Table 6.2. Different wire width and spacing cause energy differences in *W-*, *B-*, and *L-Wires*, while smaller and fewer repeaters

Table 6.2. Wire delay and relative energy parameters for each RC-based wire.

WIRE IMPLEMENTATION	RELATIVE DELAY	CROSSBAR LATENCY	RING HOP LATENCY	RELATIVE LEAKAGE	RELATIVE DYNAMIC
W-WIRES	1.0			1.00	1.00
PW-WIRES	1.2	3 CYCLES	6 CYCLES	0.30	0.30
B-WIRES	0.8	2 CYCLES	4 CYCLES	0.55	0.58
L-WIRES	0.3	1 CYCLE	2 CYCLES	0.79	0.84

cause a 70% energy decrease between *W-Wires* and *PW-Wires*. Chang et al. [31] report a factor of three reduction in energy consumption by employing transmission line technology. Thus, low-bandwidth transfers effected on *L-Wires* can not only improve performance, but also reduce energy consumption. For the evaluations, were restriced to RC-based *L-Wires*.

This analysis does not model the power consumed within the schedulers for the inter-cluster network. Heterogeneity will likely result in negligible power overhead in the schedulers while comparing networks with equal issue bandwidth.

6.4.3 Behavior of L-Wires

It is first examined how *L-Wires* enable the optimizations described in Section 6.3. Figure 6.2 shows IPCs for SPEC2k programs for two 4-cluster systems. The first is the baseline organization that has only one interconnect layer comprised entirely of *B-Wires*. Each link to a cluster can transfer 72 bits of data and tag in each direction, while the link to the data cache can transfer 144 bits in each direction. In the second 4-cluster system shown in Figure 6.2, the baseline interconnect is augmented with another metal layer that is comprised entirely of *L-Wires*. Each link to a cluster can transfer 18 bits of data and tag in each direction and the link to the cache can transfer 36 bits in each direction. The *L-Wires* are employed to send the LS bits of a load or store effective address, for the transfer of narrow bit-width data, and for the transfer of branch mispredict signals. It is seen that overall performance improves by only 4.2%, while comparing

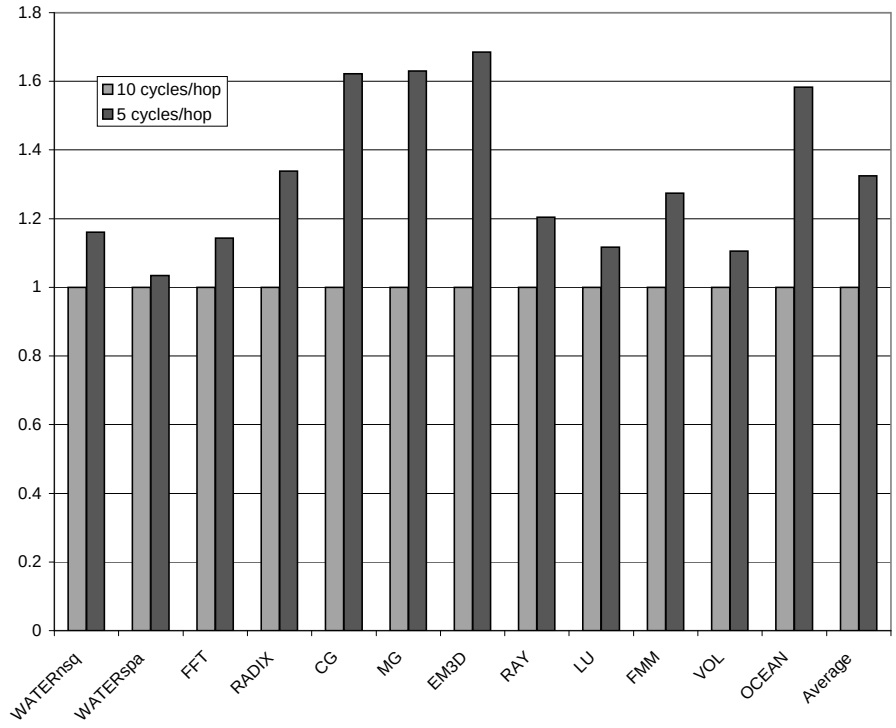


Figure 6.2. IPCs for the baseline 4-cluster partitioned architecture employing one layer of *B-Wires* and for a partitioned architecture employing one layer of *B-Wires* and one layer of *L-Wires*. The *L-Wires* transmit narrow bit-width data, branch mispredict signals, and LS bits of load/store addresses.

the arithmetic mean (AM) of IPCs[2]. It is observed that the novel cache pipeline, the transfer of narrow bit-width operands, and the transfer of branch mispredict signals, contributed equally to the performance improvement. In this particular processor model, the transfer on *L-Wires* can save at most a single cycle, yielding a modest performance improvement. Considering that the proposed pipeline entails nontrivial complexity to determine operand bit-widths and compare multiple tags at the LSQ, it is believed that the performance improvement is likely not worth the design effort. However, as listed below, there may be other scenarios where *L-Wires* can yield significant benefits.

If future technology points are more wire constrained, the latency gap between *B-Wires* and *L-Wires* widens. If latencies that are twice as much as those listed in

[2]The AM of IPCs represents a workload where every program executes for an equal number of cycles [60].

Table 6.2 is assumed, the performance improvement by adding an interconnect layer comprised of *L-Wires* is 7.1%. As transistor budgets increase, high-performance processors may employ as many as 16 clusters. Such an aggressive architecture can not only improve thread-level parallelism (TLP), but it can also improve single-thread performance for high-ILP programs. For the base processor model with a single interconnect layer comprised of *B-Wires*, the improvement in single-thread IPC by moving from 4 to 16 clusters for the 23 SPEC-2k programs is 17%. Again, the single-thread performance improvement in moving from 4 to 16 clusters likely does not warrant the complexity increase. However, if processors are going to employ many computational units for TLP extraction, the complexity entailed in allowing a single thread to span across 16 clusters may be tolerable. For such a wire-delay-constrained 16-cluster system, the performance improvement by adding a metal layer with *L-Wires* is 7.4%. As the subsequent tables shall show, there are other processor and interconnect models where the addition of an *L-Wire* interconnect layer can improve performance by more than 10%. It is possible that the novel cache pipeline may yield higher benefits for ISAs with fewer registers that may have more loads and stores. Only 14% of all register traffic on the inter-cluster network are comprised of integers between 0 and 1023. More complex encoding schemes might be required to take additional benefit of *L-Wires*. It is also possible that there are other mechanisms to exploit low-latency low-bandwidth wires that may be more complexity-effective. For example, such wires can be employed to fetch critical words from the L2 or L3.

6.4.4 Heterogeneous Interconnect Choices

The above evaluation shows performance improvements by the addition of a metal layer comprised entirely of *L-Wires*. This helps gauge the potential of *L-Wires* to reduce the cost of long wire latencies, but is not a fair comparison because of the difference in the number of metal layers. In this subsection, it is attempted to evaluate the best use of available metal area. It is started by evaluating processor models that only have enough metal area per link to each

cluster to accommodate either 144 *B-Wires*, or 288 *PW-Wires*, or 36 *L-Wires* (the link to the data cache has twice this metal area). Our base processor ($Model - I$) that effects one transfer in and out of each cluster on *B-Wires* is an example of such a processor model. The processors that have twice and thrice as much metal area are then examined, allowing more interesting combinations of heterogeneous wires. Table 6.3 summarizes the performance and energy characteristics of interesting heterogeneous interconnect organizations for a system with four clusters. All values in Table 6.3 except IPC are normalized with respect to the values for $Model - I$. ED^2 is computed by taking the product of total processor energy and the square of the number of cycles to execute 100M instructions. Total processor energy assumes that interconnect energy accounts for 10% of total chip energy in $Model - I$ and that leakage and dynamic energy are in the ratio 3:7 for $Model - I$. Table 6.3 also shows ED^2 when assuming that interconnect energy accounts for 20% of total chip energy.

First processor models that employ as much metal area as $Model - I$ are examined. The only alternative interconnect choice that makes sense is one that employs 288 *PW-Wires* for each link to a cluster ($Model - II$). A heterogeneous interconnect that consumes 1.5 times as much metal area as $Model - I$ is also evolu-

Table 6.3. Heterogeneous interconnect energy and performance for 4-cluster systems. All values (except IPC) are normalized with respect to $Model - I$. ED^2 is computed by multiplying total processor energy by square of executed cycles. 10% and 20% refer to the contribution of interconnect energy to total processor energy in $Model - I$.

Model	Description of each link	Relative Metal Area	IPC	Relative interconnect dyn-energy	Relative interconnect lkg-energy	Relative Processor Energy (10%)	Relative ED^2 (10%)	Relative ED^2 (20%)
$Model - I$	144 *B-Wires*	1.0	0.95	100	100	100	100	100
$Model - II$	288 *PW-Wires*	1.0	0.92	52	112	97	103.4	100.2
$Model - III$	144 *PW-Wires*, 36 *L-Wires*	1.5	0.96	61	90	97	95.0	92.1
$Model - IV$	288 *B-Wires*	2.0	0.98	99	194	103	96.6	99.2
$Model - V$	144 *B-Wires*, 288 *PW-Wires*	2.0	0.97	83	204	102	97.8	99.6
$Model - VI$	288 *PW-Wires*, 36 *L-Wires*	2.0	0.97	61	141	99	94.4	93.0
$Model - VII$	144 *B-Wires*, 36 *L-Wires*	2.0	0.99	105	130	101	93.3	94.5
$Model - VIII$	432 *B-Wires*	3.0	0.99	99	289	106	97.2	102.4
$Model - IX$	288 *B-Wires*, 36 *L-Wires*	3.0	1.01	105	222	104	92.0	95.5
$Model - X$	144 *B-Wires*, 288 *PW-Wires*, 36 *L-Wires*	3.0	1.00	82	233	103	92.7	95.1

ated by employing 144 *PW-Wires* and 36 *L-Wires* to each cluster ($Model - III$). From Table 6.3, it is observed that only employing slower *PW-Wires* ($Model - II$) degrades IPC and increases ED^2, in spite of the increased bandwidth. $Model - III$ with *PW-Wires* and *L-Wires* allows a combination of high performance and low energy. Most transfers happen on *PW-Wires*, resulting in 30% savings in interconnect energy dissipation, while *L-Wires* enable the optimizations described in Section 6.3 and boost performance back up to that with the baseline interconnect ($Model - I$). Thus, in terms of overall processor ED^2, the heterogeneous interconnect allows a 5% improvement, although at an area cost.

Next, processor models that have twice as much metal area per link as $Model - I$ are evaluated. $Model - IV$ accommodates 288 *B-Wires* in each link to a cluster, while $Model - V$ represents a heterogeneous interconnect that employs 144 *B-Wires* and 288 *PW-Wires*. In $Model - V$, data is assigned to *PW-Wires* according to the criteria discussed in Section 6.3. The higher latency of *PW-Wires* causes only a slight performance degradation of 1% compared to $Model - IV$. This is partly because the criteria are effective at identifying latency insensitive data transfers and partly because *PW-Wires* reduce overall contention by 14%. 36% of all data transfers happen on energy-efficient wires, leading to energy savings when compared with $Model - IV$. $Model - VI$ improves on energy and ED^2 by sending all of its traffic on *PW-Wires* and using *L-Wires* to offset the performance penalty. Finally, $Model - VII$ represents the high-performance option in this class, by employing *B-Wires* and *L-Wires*, yielding a decrease in ED^2 in spite of an increase in overall energy consumption. Thus, the interconnects with the best ED^2 employ combinations of different wires and not a homogeneous set of wires.

Finally, interesting designs that have enough area per link to a cluster to accommodate 432 *B-Wires* ($Model - VIII$) are evaluated. The high-performance option in this class ($Model - IX$) employs 288 *B-Wires* and 36 *L-Wires*, while the low-power option ($Model - X$) accommodates *B-*, *PW-*, and *L-* wires. While there is little performance benefit to be derived from having thrice as much metal area

as $Model - I$, it is interesting to note that heterogeneous interconnects continue to yield the best ED^2 values.

The evaluation is repeated on a 16-cluster system that is likely to be more sensitive to interconnect design choices. Table 6.4 summarizes the IPC, processor energy, and ED^2 values while assuming that interconnect energy accounts for 20% of total processor energy. Up to 11% reductions in ED^2 can be observed by employing heterogeneous interconnects.

In summary, our results indicate that heterogeneous wires have the potential to improve performance and energy characteristics, as compared to a baseline approach that employs homogeneous wires. Overall processor ED^2 reductions of up to 8% for 4-cluster systems and 11% for 16-cluster systems is seen by employing energy-efficient and low-latency wires. As previously discussed, the improvements can be higher in specific processor models or if transmission line technology becomes feasible.

There are clearly some nontrivial costs associated with the implementation of a heterogeneous interconnect, such as pipeline modifications, demultiplexing in the send buffers, logic to identify narrow bit-widths and network load imbalance, etc.

Table 6.4. Heterogeneous interconnect energy and performance for 16-cluster systems where interconnect energy contributes 20% of total processor energy in $Model - I$. All values (except IPC) are normalized with respect to $Model - I$.

Model	Description of each link	IPC	Relative Processor Energy (20%)	Relative ED^2 (20%)
$Model - I$	144 *B-Wires*	1.11	100	100
$Model - II$	288 *PW-Wires*	1.05	94	105.3
$Model - III$	144 *PW-Wires*, 36 *L-Wires*	1.11	94	93.6
$Model - IV$	288 *B-Wires*	1.18	105	93.1
$Model - V$	144 *B-Wires*, 288 *PW-Wires*	1.15	104	96.5
$Model - VI$	288 *PW-Wires*, 36 *L-Wires*	1.13	97	93.2
$Model - VII$	144 *B-Wires*, 36 *L-Wires*	1.19	102	88.7
$Model - VIII$	432 *B-Wires*	1.19	111	96.2
$Model - IX$	288 *B-Wires*, 36 *L-Wires*	1.22	107	88.7
$Model - X$	144 *B-Wires*, 288 *PW-Wires*, 36 *L-Wires*	1.19	106	91.9

The above results demonstrate the high potential of such an approach, necessitating a more careful examination of whether these overheads are tolerable.

6.5 Related Work

Here, other related work that has not already been cited in context is mentioned. Austin and Sohi [8] propose mechanisms to overlap cache indexing with effective address calculation. These mechanisms differ from the proposed cache pipeline in the following two major aspects: (i) they serve to hide the cost of deep pipelines and arithmetic computations, not wire delays, (ii) they employ prediction techniques.

A recent study by Citron [35] examines entropy within data being transmitted on wires and identifies opportunities for compression. The author suggests that if most traffic can be compressed, the number of wires can scale down, allowing each wire to be fatter. Unlike this proposal, the author employs a single interconnect to transfer all data and not a hybrid interconnect with different latency/bandwidth/power characteristics. A study by Loh [77] exploits narrow bitwidths to execute multiple instructions on a single 64-bit datapath. Performance improves because the effective issue width increases in some cycles. Brooks and Martonosi [25] show that in a 64-bit architecture, roughly 50% of all integer ALU operations in SPEC95-Int have both operands with bit-widths less than 16 bits. In their study, this property was exploited to reduce power consumption in integer ALUs.

The recent paper by Beckmann and Wood [20] on Transmission Line Caches is the only study that exploits low latency transmission lines at the microarchitectural level. Taylor et al. [113] define the inter-cluster communication fabric as a *Scalar Operand Network* and provide a detailed analysis of the properties of such a network and the effect of these properties on ILP extraction. Wang et al. [122] examine power consumed within on-chip interconnects, with a focus on the design of router microarchitectures. No prior architectural work has examined trade-offs in wire characteristics and the design of microarchitectures to exploit a variety of wire implementations. Thus, to the best of the knowledge, this is the first proposal of wire management at the microarchitectural level.

6.6 Summary

The design of the inter-cluster interconnect has a significant impact on overall processor energy and performance. A single wire implementation is unable to simultaneously meet the high bandwidth, low-latency, and low-energy requirements of such an interconnect. A heterogeneous interconnect that consists of wires with different properties can better meet the varying demands of inter-cluster traffic.

The chapter discusses three key contributions:

- It is shown that a low-latency low-bandwidth network can be effectively used to hide inter-cluster wire latencies and improve performance.
- It is shown that a high-bandwidth low-energy network and an instruction assignment heuristic are effective at reducing contention cycles and total processor energy.
- A comprehensive evaluation of different combinations of heterogeneous interconnects is carried out and shows that by selecting the right combination of wires, total processor ED^2 can be reduced by up to 11%, compared to a baseline processor with homogeneous interconnects.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

The dissertation proposes microarchitectural wire management and demonstrates that exposing wire properties to architects has the potential to improve both performance and power characteristics of future processors. In Chapter 2, it is first shown that a number of different wire implementations are possible in a network. For example, by tuning the wire width and spacing, wires with varying latency and bandwidth properties can be designed. Similarly, by tuning repeater size and spacing, wires with varying latency and energy properties can be designed. Further, as interconnect technology develops, transmission lines may become a commercial reality, enabling very low latency for very low-bandwidth communication. Data transfers on the on-chip network also have different requirements – some transfers benefit from a low latency network, others benefit from a high bandwidth network, and yet others are latency insensitive. To take advantage of VLSI techniques and to better match interconnect design to communication requirements, a *heterogeneous interconnect* is proposed, where every link consists of wires that are optimized for either latency, energy, or bandwidth. In the subsequent chapters, novel mechanisms are discussed that can take advantage of these interconnect choices to improve performance *and* reduce energy consumption of on-chip caches. In Chapter 3, the role of interconnects in designing large caches is demonstrated. With the insights gained from Chapter 3, Chapter 4, and Chapter 5, the techniques to exploit heterogeneous interconnections to improve cache performance and reduce inter-core communication overhead is discussed. Chapter 6 focuses on accelerating L1 accesses in a clustered architecture. Thus, the dissertation studies every aspect

of communication happening within a cache hierarchy and proposes techniques to improve their efficiency.

- **Cache Design Methodology**

  In Chapter 3, a novel methodology is proposed that identifies a cache organization that strikes the right balance between network parameters and bank parameters. It is shown that a combined design space exploration of bank and network parameters yields an organization that performs 114% better in terms of performance and consumes 50% less power compared to traditional models. This is the first body of work that highlights the role of interconnect parameters in cache access and demonstrates the importance of comprehensive interconnect-bank design space exploration in estimating cache parameters. Following the success of this approach, to facilitate future architecture research on memory hierarchy design, an open source cache modeling tool called CACTI with new network parameters is significantly enhanced and incorporated in this methodology to model large caches. This work appeared in MICRO 2007 [92].

- **Architecting Interconnection Network for Large Caches**

  Future processors are capable of having large level 2 or level 3 caches. While large caches are effective in reducing cache miss rates, the interconnect overhead associated with large structures severely limits the performance benefit of large caches. In Chapter 4, three novel pipelining techniques to address this problem and improve cache performance are proposed. All the proposed optimizations are based on four key observations: 1) In a typical cache access, a major portion of the access requires just a few lower order bits to complete their operation. The remaining higher order bits are used only during the final tag match. Hence, within a single address request, a subset of address bits is more latency critical compared to the rest of the address. 2) For a majority of the cache accesses, a partial tag match that employs a subset of tag bits is sufficient to predict the state of a cache block. 3) On-chip routers employed in a NUCA cache model introduce a nontrivial overhead in terms of both delay and power for cache accesses. 4)

Address and data networks have different requirements: the address network requires low-latency while the data network needs more bandwidth.

The first optimization, *early-lookup*, leverages the fast but low-bandwidth wires to send the critical lower order bits to initiate the lookup while the rest of the address transfer happens in parallel. The second optimization, *aggressive-lookup*, extends the above technique by exploiting the accuracy of partial tag match to eliminate the need for the full address transfer. To further optimize the low-latency wiring in the address network, a novel hybrid topology is proposed; instead of a grid network, multiple shared buses are connected through point-to-point links. This not only reduces router overhead, but also takes advantage of the ability of the low-latency wires to travel longer distances in a single cycle, and continues to support the relatively low bandwidth demands of the address network. Thus, the hybrid model introduces three forms of heterogeneity: (i) different types of wires are used in address and data networks, (ii) different topologies are used for address and data networks, (iii) the address network uses different architectures. The evaluation of these ideas has appeared in ISCA '07 [91].

- **Heterogeneous Interconnect for Coherence Traffic**

  Chip multiprocessors have an added complexity of maintaining coherence among different L1 or L2 caches. Coherence operations entail frequent communication between different caches and these messages have diverse need in terms of latency and bandwidth. In Chapter 5, the opportunities are identified to improve performance and reduce power by exploiting hop imbalances in coherence messages and carefully mapping the coherence traffic to a heterogeneous network. This work appeared in ISCA 2006 [32].

- **Heterogeneous Interconnect for Clustered Architectures**

  In Chapter 6, the techniques to improve L1 cache access speed and reduce inter-cluster communication overhead is discussed. The proposed optimizations not only improved performance but also reduced power consumption leading to a

significant improvement in ED^2. The evaluation of heterogeneous interconnect in clustered architectures appeared in HPCA 2005 [13].

Thus, microarchitectural wire management is applied to four different scenarios, all pertaining to wire-limited cache access within future multi-cores. These results show that this approach has the potential to greatly improve power and performance characteristics of future processors with minimal increases in complexity. This gives great confidence in the thesis statement and it is believed that architecture design will benefit greatly if wire properties are within its control.

7.1 Impact of the Dissertation

Since the work on heterogeneous interconnect is published, a number of other groups have also considered this technology [48, 94, 102, 105]. Rochecouste et al. [105] proposed a new partition scheme for clustered architectures and employed L-wires to accelerate media benchmarks. Nagpal et al. [94] proposed a scheduling algorithm that leverages heterogeneous interconnect to reduce communication overhead in clustered architecture. Flores et al. [48] employed an interconnect that consists of just L-wires and PW-wires and proposed "Reply partitioning" technique to improve communication efficiency. Ramani et al. [102] studied the application of the heterogeneous interconnect in a graphics processor. CACTI 6.0 has been widely used by many leading research groups to evaluate proposals related to on-chip caches and interconnection networks.

7.2 Future Work

- **Interconnection Network for Many-Core Processors**

  The efficient execution of multithreaded programs on future multi-cores will require fast inter-core communication. The current work can be extended to explore the potential of upcoming interconnect technologies such as optical interconnects, on-chip wireless data transmission, and variants of differential signaling in addressing the challenges of future many-core processors. For example,

low-swing wires are traditionally projected as a powerful medium for very low power data transfers. However, a wide spectrum of low-swing wires with various power/delay characteristics are possible by adjusting the drive voltage of the low-swing driver. This gives a unique opportunity to dynamically adapt the interconnection network to handle burst traffic, alleviate contention issues, and fine tune routing mechanisms based on the workload behavior.

- **Cache Hierarchy**

  As per industry projections, Moore's law will continue to hold at least for the next decade. Processors with hundreds of processing cores on a single die are just a few years away. Other promising technologies such as 3D stacking enable us to have multiple megabytes of on-chip caches with the possibility of dedicating an entire die for on-chip storage. With all the inter-core communications happening through the cache hierarchy, the design of an optimal cache organization is crucial to the success of many-core processors.

  Cache hierarchy design for next generation processors still remains an open issue. Workloads with a small working set favor a deep cache hierarchy with private L2 caches while parallel programs with large footprints prefer shallow cache model with a large shared cache. A systematic tool to identify an optimal cache hierarchy depth and interconnect topology will be invaluable to future architecture research.

  An inherent trade-off exists between the cost of coherence operations and the complexity of interconnection network. A shallow cache model needs a number of on-chip routers to connect various banks. However, the energy and performance cost of coherence operations are relatively small. On the other hand, a deep cache hierarchy with a shared cache in the last level requires very few banks and hence, fewer on-chip routers. However, every coherence operation should traverse multiple levels of caches leading to a high performance and power cost. Proper analysis of this trade-off can shed light on the design of cache models for next generation processors. To better suit the program behavior and to maximize

performance, as an extension to the framework, it may also be possible to consider reconfiguration of cache models at runtime.

- **Interconnect Aware Transactional Memory**

  The evolution of multi-core has put significant onus on compilers and program developers. Traditional parallel programming techniques are too complex for ubiquitous adoption. At the same time, the success of future CMPs greatly depends on the ability to develop massively parallel programs. Transactional Memory (TM) that gives the notion of atomic execution of critical section to programmers without necessarily serializing the code is a promising step towards simplifying the job for developers. The past few years have seen a number of proposals from various research groups on different flavors of Hardware Transactional Memory (HTM). It may be possible to leverage interconnect technologies to address the limitations of existing HTMs and reduce the overhead of hardware transactions. Seamless integration and reconfiguration of caches and HTM buffers, novel network topologies for fast inter buffer communications, novel interconnection networks that dynamically adjust to the transaction needs are some of the interesting areas to explore.

REFERENCES

[1] SGI Altix 3000 Configuration. http://www.sgi.com/products/servers/ altix/configs.html.

[2] M. E. Acacio, J. Gonzalez, J. M. Garcia, and J. Duato. The Use of Prediction for Accelerating Upgrade Misses in CC-NUMA Multiprocessors. In *Proceedings of PACT-11*, 2002.

[3] V. Agarwal, M. Hrishikesh, S. Keckler, and D. Burger. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. In *Proceedings of ISCA-27*, pages 248–259, June 2000.

[4] A. Aggarwal and M. Franklin. An Empirical Study of the Scalability Aspects of Instruction Distribution Algorithms for Clustered Processors. In *Proceedings of ISPASS*, 2001.

[5] A. Aggarwal and M. Franklin. Hierarchical Interconnects for On-Chip Clustering. In *Proceedings of IPDPS*, April 2002.

[6] Arizona State University. Predictive Technology Model. http://www.eas.asu.edu/˜ptm.

[7] S. I. Association. International Technology Roadmap for Semiconductors 2005. http://public.itrs.net/Links/2005ITRS/Home2005.htm.

[8] T. M. Austin and G. Sohi. Zero-Cycle Loads: Microarchitecture Support for Reducing Load Latency. In *Proceedings of MICRO-28*, November 1995.

[9] H. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley, 1990.

[10] H. Bakoglu and J. Meindl. A System-Level Circuit Model for Multi- and Single-Chip CPUs. In *Proceedings of ISSCC*, 1987.

[11] R. Balasubramonian. Cluster Prefetch: Tolerating On-Chip Wire Delays in Clustered Microarchitectures. In *Proceedings of ICS-18*, June 2004.

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

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

[14] K. Banerjee and A. Mehrotra. A Power-optimal Repeater Insertion Methodology for Global Interconnects in Nanometer Designs. *IEEE Transactions on Electron Devices*, 49(11):2001–2007, November 2002.

[15] K. Banerjee, A. Mehrotra, A. Sangiovanni-Vincentelli, and C. Hu. On Thermal Effects in Deep Submicron VLSI Interconnects. In *Proceedings of the Design Automation Conference*, pages 885–891, 1999.

[16] A. Baniasadi and A. Moshovos. Instruction Distribution Heuristics for Quad-Cluster, Dynamically-Scheduled, Superscalar Processors. In *Proceedings of MICRO-33*, pages 337–347, December 2000.

[17] P. Bannon. Alpha 21364: A scalable single-chip SMP. *Microprocessor Forum*, October 1998.

[18] R. Barua, W. Lee, S. Amarasinghe, and A. Agarwal. Maps: A Compiler-Managed Memory System for Raw Machines. In *Proceedings of ISCA-26*, May 1999.

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

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

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

[22] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill, and D. A. Wood. Multicast Snooping: A New Coherence Method using a Multicast Address Network. *SIGARCH Comput. Archit. News*, pages 294–304, 1999.

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

[24] F. A. Briggs, M. Cekleov, K. Creta, M. Khare, S. Kulick, A. Kumar, L. P. Looi, C. Natarajan, S. Radhakrishnan, and L. Rankin. Intel 870: A building block for cost-effective, scalable servers. *IEEE Micro*, 22(2):36–47, 2002.

[25] D. Brooks and M. Martonosi. Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance. In *Proceedings of HPCA-5*, January 1999.

[26] D. Burger and T. Austin. The Simplescalar Toolset, Version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, June 1997.

[27] A. Caldwell, Y. Cao, A. Kahng, F. Koushanfar, H. Lu, I. Markov, M. Oliver, D. Stroobandt, and D. Sylvester. GTX: The MARCO GSRC Technology Extrapolation System. In *Proceedings of DAC*, 2000.

[28] R. Canal, J. M. Parcerisa, and A. Gonzalez. Dynamic Cluster Assignment Mechanisms. In *Proceedings of HPCA-6*, pages 132–142, January 2000.

[29] A. Cester, L. Bandiera, S. Cimino, A. Paccagnella, and G. Ghidini. Incidence of Oxide and Interface Degradation on MOSFET Performance. In *Proceedings of Biennial Conference on Insulating Films on Semiconductors)*, pages 66–70, May 2004.

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

[31] R. Chang, N. Talwalkar, C. Yue, and S. Wong. Near Speed-of-Light Signaling Over On-Chip Electrical Interconnects. *IEEE Journal of Solid-State Circuits*, 38(5):834–838, May 2003.

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

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

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

[35] D. Citron. Exploiting Low Entropy to Reduce Wire Delay. *IEEE Computer Architecture Letters*, vol.2, January 2004.

[36] J. Collins and D. Tullsen. Clustered Multithreaded Architectures – Pursuing Both IPC and Cycle Time. In *Proceedings of the 18th IPDPS*, April 2004.

[37] Corporate Institute of Electrical and Electronics Engineers, Inc. Staff. *IEEE Standard for Scalable Coherent Interface, Science: IEEE Std. 1596-1992.* 1993.

[38] A. Cox and R. Fowler. Adaptive Cache Coherency for Detecting Migratory Shared Data. pages 98–108, May 1993.

[39] D. E. Culler and J. P. Singh. *Parallel Computer Architecture: a Hardware/software Approach.* Morgan Kaufmann Publishers, Inc, 1999.

[40] W. Dally. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2), March 1992.

[41] W. Dally and J. Poulton. *Digital System Engineering.* Cambridge University Press, Cambridge, UK, 1998.

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

[43] A. Deutsch. Electrical Characteristics of Interconnections for High-Performance Systems. *Proceedings of the IEEE*, 86(2):315–355, Feb 1998.

[44] J. Eble. A Generic System Simulator (Genesys) for ASIC Technology and Architecture Beyond 2001. In *Proceedings of 9th IEEE International ASIC Conference*, 1996.

[45] N. Eisley, L.-S. Peh, and L. Shang. In-Network Cache Coherence. In *Proceedings of MICRO-39*, December 2006.

[46] J. Emer, M. D. Hill, Y. N. Patt, J. J. Yi, D. Chiou, and R. Sendag. Single-Threaded vs. Multithreaded: Where Should We Focus? In *IEEE Micro*, 2007.

[47] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic. The Multicluster Architecture: Reducing Cycle Time through Partitioning. In *Proceedings of MICRO-30*, pages 149–159, December 1997.

[48] A. Flores, J. L. Aragon, and M. E. Acacio. Inexpensive Implementations of Set-Associativity. In *Proceedings of HiPC*, 2007.

[49] M. Galles and E. Williams. Performance Optimizations, Implementation, and Verification of the SGI Challenge Multiprocessor. In *HICSS (1)*, pages 134–143, 1994.

[50] G. Gerosa and et al. A 2.2 W, 80 MHz Superscalar RISC Microprocessor. *IEEE Journal of Solid-State Circuits*, 29(12):1440–1454, December 1994.

[51] B. M. Geuskins. *Modeling the Influence of Multilevel Interconnect on Chip Performance*. PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, 1997.

[52] E. Gibert, J. Sanchez, and A. Gonzalez. Flexible Compiler-Managed L0 Buffers for Clustered VLIW Processors. In *Proceedings of MICRO-36*, December 2003.

[53] R. Ho. *On-Chip Wires: Scaling and Efficiency*. PhD thesis, Stanford University, August 2003.

[54] R. Ho, K. Mai, and M. Horowitz. The Future of Wires. *Proceedings of the IEEE*, Vol.89, No.4, April 2001.

[55] R. Ho, K. Mai, and M. Horowitz. Managing Wire Scaling: A Circuit Prespective. *Interconnect Technology Conference*, pages 177–179, June 2003.

[56] J. Huh, J. Chang, D. Burger, and G. S. Sohi. Coherence Decoupling: Making Use of Incoherence. In *Proceedings of ASPLOS-XI*, pages 97–106, 2004.

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

[58] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A NUCA Substrate for Flexible CMP Cache Sharing. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 31–40, New York, NY, USA, 2005. ACM Press.

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

[60] L. John. More on Finding a Single Number to Indicate Overall Performance of a Benchmark Suite. *ACM Computer Architecture News*, 32(1), March 2004.

[61] U. Kapasi, W. Dally, S. Rixner, J. Owens, and B. Khailany. The Imagine Stream Processor. In *Proceedings of ICCD*, September 2002.

[62] S. Keckler and W. Dally. Processor Coupling: Integrating Compile Time and Runtime Scheduling for Parallelism. In *Proceedings of ISCA-19*, pages 202–213, May 1992.

[63] R. Kessler. The Alpha 21264 Microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.

[64] R. E. Kessler, R. Jooss, A. Lebeck, and M. Hill. Inexpensive Implementations of Set-Associativity. In *Proceedings of ISCA-16*, 1989.

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

[66] H.-S. Kim and J. Smith. An Instruction Set and Microarchitecture for Instruction Level Distributed Processing. In *Proceedings of ISCA-29*, May 2002.

[67] K. Krewell. UltraSPARC IV Mirrors Predecessor: Sun Builds Dualcore Chip in 130nm. *Microprocessor Report*, pages 1,5–6, Nov. 2003.

[68] A. Kumar, L. Peh, P. Kundu, and N. K. Jha. Express Virtual Channels: Towards the Ideal Interconnection Fabric. In *Proceedings of ISCA*, June 2007.

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

[70] A.-C. Lai and B. Falsafi. Memory Sharing Predictor: The Key to a Speculative Coherent DSM. In *Proceedings of ISCA-26*, 1999.

[71] A.-C. Lai and B. Falsafi. Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction. In *Proceedings of ISCA-27*, pages 139–148, 2000.

[72] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proceedings of ISCA-24*, pages 241–251, June 1997.

[73] A. R. Lebeck and D. A. Wood. Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors. In *Proceedings of ISCA-22*, pages 48–59, 1995.

[74] K. M. Lepak and M. H. Lipasti. Temporally Silent Stores. In *Proceedings of ASPLOS-X*, pages 30–41, 2002.

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

[76] J. Li, J. F. Martinez, and M. C. Huang. The Thrifty Barrier: Energy-Aware Synchronization in Shared-Memory Multiprocessors. In *HPCA '04: Proceedings of the 10th International Symposium on High Performance Computer Architecture*, page 14, Washington, DC, USA, 2004. IEEE Computer Society.

[77] G. Loh. Exploiting Data-Width Locality to Increase Superscalar Execution Bandwidth. In *Proceedings of MICRO-35*, November 2002.

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

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

[80] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *IEEE Computer*, 35(2):50–58, February 2002.

[81] M. Mamidipaka and N. Dutt. eCACTI: An Enhanced Power Estimation Model for On-Chip Caches. Technical Report CECS Technical Report 04-28, University of California, Irvine, September 2004.

[82] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood. Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset. *Computer Architecture News*, 2005.

[83] M. M. K. Martin, M. D. Hill, and D. A. Wood. Token Coherence: Decoupling Performance and Correctness. In *Proceedings of ISCA-30*, 2003.

[84] D. Matzke. Will Physical Scalability Sabotage Performance Gains? *IEEE Computer*, 30(9):37–39, September 1997.

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

[86] M. Minzuno, K. Anjo, Y. Sume, M. Fukaishi, H. Wakabayashi, T. Mogami, T. Horiuchi, and M. Yamashina. Clock Distribution Networks with On-Chip Transmission Lines. In *Proceedings of the IEEE International Interconnect Technology Conference*, pages 3–5, 2000.

[87] M. L. Mui, K. Banerjee, and A. Mehrotra. A Global Interconnect Optimization Scheme for Nanometer Scale VLSI With Implications for Latency, Bandwidth, and Power Dissipation. *IEEE Transactions on Electronic Devices*, Vol.51, No.2, February 2004.

[88] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 Network Architecture. In *IEEE Micro*, volume 22, 2002.

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

[90] R. Mullins, A. West, and S. Moore. Low-Latency Virtual-Channel Routers for On-Chip Networks. In *Proceedings of ISCA-31*, May 2004.

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

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

[93] R. Nagarajan, K. Sankaralingam, D. Burger, and S. Keckler. A Design Space Evaluation of Grid Processor Architectures. In *Proceedings of MICRO-34*, pages 40–51, December 2001.

[94] R. Nagpal and Y. N. Srikant. Exploring Energy-Performance Trade-offs for Heterogeneous Interconnect in Clustered VLIW Processors. In *Proceedings of High Performance Computing*, 2006.

[95] N. Nelson, G. Briggs, M. Haurylau, G. Chen, H. Chen, D. Albonesi, E. Friedman, and P. Fauchet. Alleviating Thermal Constraints while Maintaining Performance Via Silicon-Based On-Chip Optical Interconnects. In *Proceedings of Workshop on Unique Chips and Systems*, March 2005.

[96] K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K.-Y. Chang. The Case for a Single-Chip Multiprocessor. In *Proceedings of ASPLOS-VII*, October 1996.

[97] J.-M. Parcerisa, J. Sahuquillo, A. Gonzalez, and J. Duato. Efficient Interconnects for Clustered Microarchitectures. In *Proceedings of PACT*, September 2002.

[98] L.-S. Peh and W. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In *Proceedings of HPCA-7*, 2001.

[99] L.-S. Peh and W. Dally. A Delay Model for Router Micro-architectures. *IEEE Micro*, 21(1):26–34, January/February 2001.

[100] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits - A Design Perspective*. Prentice-Hall, 2nd edition, 2002.

[101] P. Racunas and Y. Patt. Partitioned First-Level Cache Design for Clustered Microarchitectures. In *Proceedings of ICS-17*, June 2003.

[102] K. Ramani, A. Ibrahim, and D. Shimizu. PowerRed: A Flexible Modeling Framework for Power Efficiency Exploration in GPUs. In *Workshop on General Purpose Processing on Graphics Processing Units*, 2007.

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

[104] G. Reinman and N. Jouppi. CACTI 2.0: An Integrated Cache Timing and Power Model. Technical Report 2000/7, WRL, 2000.

[105] O. Rochecouste, G. Pokam, and A. Seznec. A Case for a Complexity-effective, Width-partitioned Microarchitecture. In *ACM Transactions on Architecture and Code Optimization(TACO)*, 2006.

[106] J. Sanchez and A. Gonzalez. Modulo Scheduling for a Fully-Distributed Clustered VLIW Architecture. In *Proceedings of MICRO-33*, pages 124–133, December 2000.

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

[108] P. Shivakumar and N. P. Jouppi. CACTI 3.0: An Integrated Cache Timing, Power, and Area Model. Technical Report TN-2001/2, Compaq Western Research Laboratory, August 2001.

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

[110] P. Stenström, M. Brorsson, and L. Sandberg. An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing. pages 109–118, May 1993.

[111] D. Sylvester and K. Keutzer. System-Level Performance Modeling with BACPAC - Berkeley Advanced Chip Performance Calculator. In *Proceedings of 1st International Workshop on System-Level Interconnect Prediction*, 1999.

[112] D. Tarjan, S. Thoziyoor, and N. Jouppi. CACTI 4.0. Technical Report HPL-2006-86, HP Laboratories, 2006.

[113] M. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal. Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures. In *Proceedings of HPCA-9*, February 2003.

[114] M. Taylor, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, J. Kim, J. Psota, A. Raraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. In *Proceedings of ISCA-31*, June 2004.

[115] J. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy. POWER4 System Microarchitecture. Technical report, IBM Server Group Whitepaper, October 2001.

[116] S. Thoziyoor, N. Muralimanohar, and N. Jouppi. CACTI 5.0. Technical Report HPL-2007-167, HP Laboratories, 2007.

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

[118] E. Tune, D. Liang, D. Tullsen, and B. Calder. Dynamic Prediction of Critical Path Instructions. In *Proceedings of HPCA-7*, pages 185–196, January 2001.

[119] V. Venkatraman, A. Laffely, J. Jang, H. Kukkamalla, Z. Zhu, and W. Burleson. NOCIC: A Spice-Based Interconnect Planning Tool Emphasizing Aggressive On-Chip Interconnect Circuit Methods. In *Proceedings of International Workshop on System Level Interconnect Prediction*, February 2004.

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

[121] H.-S. Wang, L.-S. Peh, and S. Malik. A Power Model for Routers: Modeling Alpha 21364 and InfiniBand Routers. volume 23, January/February 2003.

[122] H.-S. Wang, L.-S. Peh, and S. Malik. Power-Driven Design of Router Microarchitectures in On-Chip Networks. In *Proceedings of MICRO-36*, December 2003.

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

[124] J. Warnock, J. Keaty, J. Petrovick, J. Clabes, C. Kircher, B. Krauter, P. Restle, B. Zoric, and C. Anderson. The Circuit and Physical Design of the POWER4 Microprocessor. *IBM Journal of Research and Development*, 46(1):27–51, Jan 2002.

[125] S. Wilton and N. Jouppi. An Enhanced Access and Cycle Time Model for On-Chip Caches. Technical Report TN-93/5, Compaq Western Research Lab, 1993.

[126] S. Wilton and N. Jouppi. An Enhanced Cache Access and Cycle Time Model. *IEEE Journal of Solid-State Circuits*, May 1996.

[127] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of ISCA-22*, pages 24–36, June 1995.

[128] T. Xanthopoulos, D. Bailey, A. Gangwar, M. Gowan, A. Jain, and B. Prewitt. The Design and Analysis of the Clock Distribution Network for a 1.2GHz Alpha Microprocessor. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 402–403, 2001.

[129] J. Yang, Y. Zhang, and R. Gupta. Frequent Value Compression in Data Caches. In *Proceedings of MICRO-33*, pages 258–265, December 2000.

[130] H. Zhang, G. Varghese, and J. Rabaey. Low-swing On-chip Signalling Techniques. *Transactions on VLSI Systems*, pages 264–272, 2000.

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