L2 Cache to Off-chip Memory Networks for Chip Multiprocessor

Carrell Killebrew

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-71
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-71.html

May 23, 2008

L2 Cache to Off-chip Memory Networks for Chip Multiprocessors

by Carrell D Killebrew

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Krste Asanovic
Research Advisor

(Date)

* * * * * * *

Professor John Kubiatowicz
Second Reader

(Date)

L2 Cache to Off-chip Memory Networks for Chip Multiprocessors

Abstract

L2 Cache to Off-chip Memory Networks for Chip Multiprocessors

by

Carrell D Killebrew
Master of Science in Computer Science

University of California at Berkeley

Professor Krste Asanovic, Research Advisor

Manycores are at the heart of the oncoming sea change in computing. Placing many simple cores on a single die is predicted to be the answer to the power and ILP walls that the dominant design trends of the past couple decades have hit. Manycores will make new workloads possible; workloads that place a greatly increased strain on off-chip memory bandwidth. Some of these have working sets that simply cannot fit inside a conventional cache. Most of these have bandwidth requirements that scale superlinearly with the number of cores; as cores are added, the per-core traffic also increases. Providing sufficient off-chip bandwidth is the engineering challenge that must be overcome to allow the full potential of manycores to be harnessed.

In recent years, manycores have seen an increasing asymmetry between the number of cores and the number of memory access points on a single die. This asymmetry causes problems for the popular mesh topology, prompting a study using a topology that better connects many cores to few memory access points.

In this thesis, I examine several modern multicore chips, identifying design trends in the off-chip memory interconnect. I analyze a modern manycore chip, Tilera's Tile64, identifying weaknesses in its L2 to DRAM network (L2DN). As an alternative, I present a network based on a tapered fat tree (TFT) topology. I then evaluate the latency and power efficiency of the TFT topology on a cycle-accurate simulator intended to model the Tile64 multiprocessor, comparing it to the mesh network that the Tile64 actually uses. I replace the original mesh network used for off-chip memory access with two TFT networks (one for memory requests, one for memory responses) and run four synthetic benchmarks modeled after those in the PARSEC suite.

Because several connections in the TFT network require global wires, I also model the multi-cycle latencies using a wire-delay model. The simulator keeps track of activity factors in each of the routers, which I combine with the Orion power models to determine the switching power of the routers. I determined that for applications with large amounts of sharing and little off-chip traffic, the TFT topology offers no significant advantage over the mesh. However, the benchmarks that exhibited large amounts of off-chip traffic completed a workload up to 33.6% faster on the TFT than the same workload on the mesh. The TFT topology also has better energy efficiency in all cases. At the cost of extra design complexity, the results suggest that the TFT topology offers better latency, throughput, and energy-efficiency overall than the mesh for memory-intensive applications running on manycore processors.

The primary contributions of this thesis are:

- The discussion of the manycore bandwidth challenge and its effects upon a modern day manycore chip, the Tile64.
- A concrete proposal to partially alleviate the effects in the context of the Tile64.
- A thorough performance and power analysis of the proposed solution.

Contents

List of Figures iv

List of Tables v

1 Introduction 1
- 1.1 Manycore Bandwidth Challenge . . . 2

2 Background and Motivation 4
- 2.1 Tile64 . . . 5

3 Tapered Fat Trees for L2DN 9
- 3.1 Tapered Fat-Tree (TFT) . . . 9
  - 3.1.1 Topology . . . 10
  - 3.1.2 Routing Function . . . 10
  - 3.1.3 Tapered Fat-Tree on TILE64 . . . 10

4 Evaluation Models 13
- 4.1 Wire Model . . . 13
  - 4.1.1 Technology Model . . . 14
  - 4.1.2 Wire Layout . . . 14
  - 4.1.3 Multi-Cycle Wire Delay . . . 16
- 4.2 Power Model . . . 16
  - 4.2.1 Router Power . . . 16
  - 4.2.2 Interconnect Power . . . 19

5 Simulation Environment 20
- 5.1 Description of Simulator Operation . . . 20
- 5.2 Benchmark Description . . . 22

6 Results 24
- 6.1 Benchmark Completion Time . . . 24
- 6.2 Power . . . 26

7 Related Work 30

8 Conclusion 32

List of Figures

1 Tile64 memory controller connections [16] 6

2 Tapered Fat-Tree topology 10
3 Tapered Fat-Tree mapped onto TILE64. The connections only represent logical connectivity. A possible physical layout is shown in Figure 5. . . 11

4 First-order repeater model 13
5 Physical layout of TFT on TILE64 17
6 Input-buffered on-chip router with matrix arbiter. 18

7 Two physically separate networks, for requests and replies 21
8 Protocol for TFT memory traffic 22

9 Completion times of benchmarks normalized to the completion time with the Mesh topology. 25
10 Average latency of packets for the blackscholes and x264 benchmarks. . 25
11 Average latency of packets for the canneal and streamcluster benchmarks. 26
12 Power dissipated within the routers of the network. 27
13 Total power dissipated in routers and interconnect. 28
14 Relative energy dissipated in routers and interconnect. 28

List of Tables

1 Multicore Survey 5

2 Model Characteristics and Constants 15
3 Interconnect Characteristics 15
4 Multi-Cycle Wire Delays ($t_{cycle} = 1.33$ ns) 16

5 PARSEC Applications Simulation Parameters 20
6 Simulation Packet Sizes 22

7 Throughput of Selected MDN Routers 26

Chapter 1

Introduction

Manycore processors, single silicon dies with many simple cores, have gained increasing attention as the computing industry has hit the combined power and ILP wall [11]. Manycores present new engineering challenges and opportunies due to their unique new architecture. The proximity of cores makes interprocessor communication much cheaper, but providing enough memory bandwidth for all these cores to function becomes a more serious problem.

As manycores have evolved from single core architectures, we can observe the evolution of the network used to connect cores to both off-chip and on-chip memory. All designs have a private L1 cache, but the L2 cache, which is shared when there are a few cores, becomes more private as cores are added. In chips with less cores, interprocessor communication usually occurs through a shared memory model, while those with more cores sometimes add explicit interprocessor networks for message passing.

Besides offering a possible solution to the problems that have stymied the uniprocessors of the past, manycores will make interesting new workloads feasible. Applications such data mining and advanced video encoding will place an increased strain on the memory system. Some of these workloads demonstrate an increasing per-core performance as more cores are added to a chip[13]. This translates into an exponential growth in memory bandwidth demand. Clearly this exponential growth has significant implications for future manycores.

In this thesis I examine several modern multicore chips, analyzing design trends in the L2 to DRAM network (L2DN). I will summarize the results of the PARSEC benchmark suite as they apply to L2DNs. I look at a modern 64-core chip, the Tile64, and use the results of the PARSEC suite to motivate a redesign of the Tile64's L2DN.

I provide a comparison of my proposal and the Tile64's current solution using detailed power analysis and simulation results.

1.1 Manycore Bandwidth Challenge

The SPLASH2 benchmarks [2] have been the standard parallel benchmarking suite for many years now, but they are now somewhat dated. Some of the algorithms it uses are now inferior to newer methods for solving the same problems. Additionally, the benchmarks it contains are aimed at HPC and graphical simulations. manycores will be used by a much more diverse set of users with a correspondingly wider array of applications. Thus, I use the PARSEC benchmarking suite and the characterization of the applications developed in [13] as the basis for my analysis.

These new applications will have a large appetite for memory bandwidth. There are 12 different applications in the PARSEC suite, and of these 12, all but three have a working set size larger than 8 MB. This is much larger than most current on-chip caches, distributed or otherwise. Even if on-chip caches were to grow, these working set sizes grow along with the data set the algorithm is working on. Applications such as data mining become more useful as the data set is increased, so I expect data set sizes to grow with time. As a result, I anticipate that the working set of at least eight of these applications will exceed future on-chip cache sizes, creating a distinct need for off-chip memory bandwidth.

The amount of off-chip memory bandwidth for half of the applications increases with the number of cores. This means that as cores are added, the bandwidth demand increases because of the core added, as well as because the cores already on the chip increase their individual demands. This results in a superlinearly increasing demand for off-chip bandwidth.

The PARSEC applications do not exhibit the same demand for interprocessor communication that they do for off-chip bandwidth. Four of the 12 exhibit an insignificant amount of sharing. Of those with true sharing, the data sets are so large that shared accesses between different processors are temporally spaced far apart. The effect is that by the time the second processor makes a shared access, the data has already been moved off-chip because of a capacity conflict. Two of the applications are parallelized in such a way that data is passed between different computational stages of the algorithm, allowing efficient data sharing. If cores were added, this on-chip traffic would scale proportionally, but so would the interprocessor network as well, presumably. None of the applications demonstrate an increasing usage of shared (or private) cache space as the number of processors scales.

The result of this will be a divergence of the requirements of the interprocessor communication network and the L2DN. Interprocessor communication networks beyond simple buses are usually chosen such that network bandwidth scales as processors are added. This will be sufficient to handle the traffic due to shared memory access and message passing, as this traffic volume grows linearly along with the number of cores. The off-chip memory bandwidth, on the other hand, will increase superlinearly as cores are added. Working set sizes will only get larger as the utility of many of these applications increases as the data set grows (and the working set grows with the data set). The additional cores will increase the efficiency of the cores already present, resulting in the superlinear demand for off-chip memory bandwidth. The off-chip bandwidth requirements may outstrip those of the on-chip communication network. As the PARSEC authors put it, "since many PARSEC workloads have high bandwidth requirements and working sets which exceed conventional caches by far, off-chip bandwidth will be [many-cores'] most severe limitation of performance. Substantial architectural improvements are necessary to allow emerging workloads to take full advantage of larger CMPs. [13]"

Chapter 2

Background and Motivation

On-chip caches have evolved as cores have been added to the chip, and this has had a direct effect on the off-chip memory interconnect. Table 1 contains a summary of the following architectural descriptions. The Intel Core2 Duo has two cores on a die, which access a shared L2 cache and off-chip memory controller. AMD's Barcelona possesses four cores, each with a private L2 cache. A crossbar connects the cores to a shared L3 cache, which interfaces with the two memory controllers. Intel's Nehalem is similar to the Barcelona, except that there are eight cores. Sun's Niagara uses a full crossbar to connect eight cores to a four-way banked, shared L2 [15]. Sun's Rock uses a hierarchecal scheme, with the 16 cores divided into four clusters[14]. A cluster shares one instruction cache and two data caches, along with a crossbar that provides access to the central crossbar. This central crossbar connects to a four-way banked, shared L2.

These processors do not have an explicit interprocessor communication network, instead relying on coherence of the shared caches to share data and synchronize. In all cases, there is a shared L2, or shared L3 if the cores have private L2's. This L3 accesses the memory upon a miss. Thus, the interprocessor communication network is also the processor to off-chip memory interconnect.

IBM's Cell processor is unique in both its cache structure (or lack thereof) and its interconnection network. The Power Processing Element (PPE) is a standard Power core with a cache hierarchy, while the 8 Synergistic Processing Elements (SPEs) possess no caches, only a 256 KB local store each. These local stores are accessed using DMA transfer initiated by the PPE or any of the SPEs. Only the owning SPE has direct access to its local store. The PPE, SPEs, and memory controller are all connected to four rings. The rings are divided into two pairs of unidirectional links, each pair traveling in the opposite direction from the other. The rings serve as both

Table 1: Multicore Survey

Name	Cores	L2	Interconnect	Off-chip Bandwidth
Intel Core2 Duo	2 (per die)	Shared	Crossbar	10.7 GB/s
AMD Barcelona	4	Private	Crossbar	10.7 GB/s
Intel Nehalem	1-8	Private	Crossbar	31.9 GB/s
Sun Niagara	8	Shared	Crossbar	25.6 GB.s
IBM Cell	8	Local Store	4 Parallel Rings	25.6 GB/s
Sun Rock	16	Shared	Hierarchical Crossbar	54.4 GB/s
Tilera Tile64	64	Private	Mesh	25.6 GB/s

the interprocessor network and the off-chip memory network. A key difference is that there is no shared cache structure between the processors and the memory controller, as a result of the streaming model the Cell has adopted.

2.1 Tile64

Tilera's Tile64 is a CMP with 64 cores[16]. I have chosen to study it, as it has the most cores of any modern commercial manycore and has enough cores to make the scaling properties of the networks start to become evident. The 64 cores are connected in an 8x8 mesh network. Mesh networks are popular because they are simple, have an efficient two-dimensional VLSI layout, and support tiling, which makes design and verification easier as well. Each tile contains a processor, the routers that support the networks, and an L2 cache.

The Tile64 has five physically separate 32 bit mesh networks that connect the processors. The mesh networks use dimension ordered routing (DOR) and credit based flit-level flow control. The routers require two cycles to route flits which are turning, and one cycle for flits continuing in the same direction. Interrouter links take one cycle to traverse. Each router input has three flits of buffering, the minimum required to support full throughput on the interrouter links.

Each mesh serves a different purpose, the two that I am concerned with are the Tile Dynamic Network (TDN) and Memory Dynamic Network (MDN). The TDN delivers interprocessor cache-to-cache transfer requests. The actual bulk data transfer returns over the MDN to prevent protocol deadlock. An alternative would be to provide another virtual channel on the TDN to avoid deadlock, but this would require buffer space beyond the minimum used for full throughput flow control. The designers of the Tile64 specifically stated that they were attempting to minimize the chip area consumed

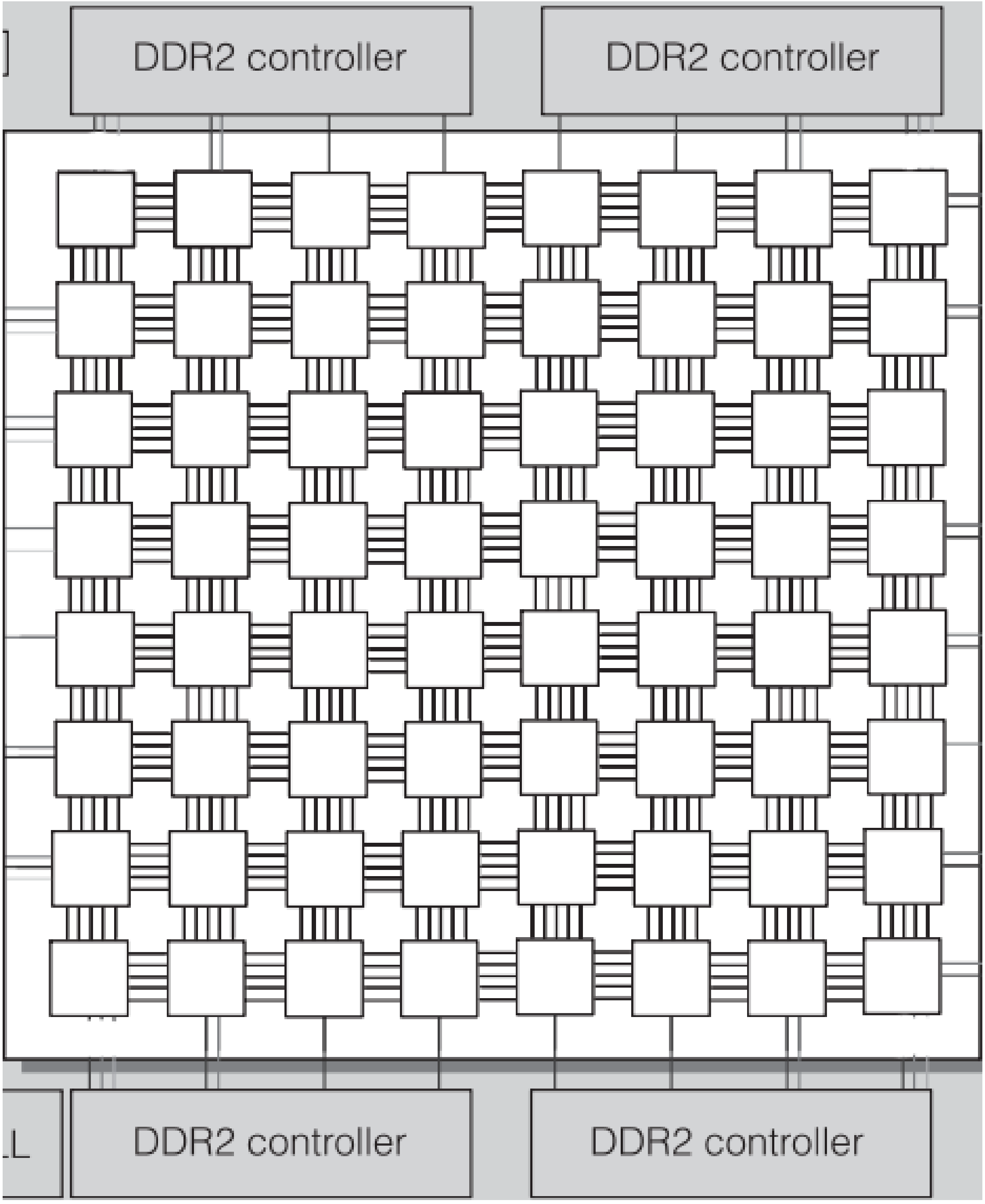

Figure 1: Tile64 memory controller connections [16]

by the networks, so they chose to multiplex the MDN. As demonstrated later by the simulation, this results in an under-utilized TDN and an over-utilized MDN.

All loads and stores that are destined for off-chip memory use the MDN. Protocol deadlock is prevented because each DRAM reserves enough buffer space to buffer a certain number of requests from each core. The cores, in turn, do not have more than this predetermined number of outstanding requests to any single DRAM. If the responses traveled a different physical or logical network, this precaution would not be necessary. Removing the limit on outstanding requests would improve throughput and save on buffer space at the DRAM, at the expense of the extra routers for a new physical network, or buffer space and complexity for a new logical network.

The connection between the memory controllers and the tiles (Figure 1) is interesting because of the mismatch between the number of memory controllers (4) and tiles (64). There are two memory controllers on the north side of the chip, and two on the south side. Each is connected to the three innermost adjacent tiles. Tiles route to the closest of the three access points (see Figure 1). The memory controller uses round-robin arbitration choose from among the three input tiles. Under a uniform traffic distribution, the three access points each see a different amount of traffic. The innermost of the three sees five times as much traffic as the middle one, and 2.5 times as much as the outer access point. This results in global unfairness because the five columns of tiles that use the innermost access point will see less throughput than the other columns of tiles.

The alternative, though, is much worse. The mesh employs DOR, with routing in the X direction followed by the Y direction. If tiles were to spread their traffic over the three access points instead of going to the closest, when the response returned from DRAM and entered the mesh, it would immediately travel in the X direction until it reached the appropriate column. Consider an example where two packets arrive, one at the middle and inner access points. Both are destined for a tile somewhere to the east, so they must take turns exiting the inner router's eastern ouput port. Add in the outer access point, and this results in horrible throughput for packets entering the mesh. Changing the routing to Y dimension followed by X would not solve anything, only cause the problem to occur as packets reached the edge and started traveling in the X dimension towards an access point. The problem is a result of the routing algorithm and the topology mismatch (four memory controllers talking to 64 cores on a mesh).

The four memory controllers support a total bandwidth of 256 bits/cycle, and the MDN has a bisection bandwidth of 1024 bits/cycle. As the MDN is shared by requests returning from memory as well as intertile bulk data transfers, this seeming 'overprovisioning' of the mesh is required to support full throughput from off-chip memory.

Chapter 3

Tapered Fat Trees for L2DN

I propose to replace the Tile64's mesh MDN with two tapered fat trees (TFTs). TFTs used as processor to processor interconnection networks have low chip area usage and power dissipation [4]. They are also superior to meshes in good workload completion times, giving good efficiency in terms of area-delay and energy-delay. Replacing the mesh MDN with TFTs means that traffic that previously used the MDN for interprocessor communication (loads from remote L2 caches) now must transit the TDN. This provides better load balancing between the two networks, another factor in improving the performance of the whole system, at the cost of adding buffer space and increasing router complexity to implement virtual channels.

3.1 Tapered Fat-Tree (TFT)

The topology shown in Figure 2 also has benefits because the top four routers map to the four memory controllers. Because all traffic on the MDN is either traveling from a processor/L2 cache to a memory controller, or visa versa, there are never any U-turns in the network. There are no packets that travel up to a middle level router, then turn around and go back down. This reduces routing complexity, eliminates crossbar connections, and as a result, lowers power consumption and area consumed. Hop count is also reduced, lowering packet latency and reducing router energy because each packet visits fewer routers. Splitting the network up into separate physical networks, one in each direction does not increase router area (it lowers it because of the simplified crossbars) [4].

All of these benefits come at a cost, however. Wires no longer travel to the nearest neighbor, meaning wires must now cross over each other. This means using

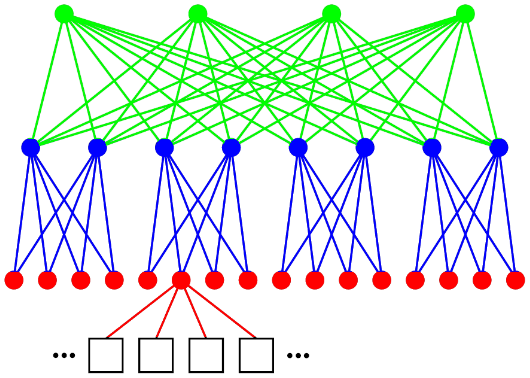

Figure 2: Tapered Fat-Tree topology

upper metal layers. Depending upon the specifics of the layout, it could require another metal layer to be added. Interrouter links are now too long to traverse in a single cycle, and so repeaters must be inserted, taking up additional area, and possibly increasing place and route complexity.

3.1.1 Topology

Figure 2 illustrates the tapered-fat tree (TFT) topology. Each of the bottom level routers (circles) connects to four processor/L2 nodes. Each router of the top level connects directly to a single memory controller.

3.1.2 Routing Function

The routing function is very simple. Packets leave their source and travel to the bottom level of routers. One of the two possible intermediate level routers is chosen based on the address of the request (addresses are interleaved across the two intermediate routers). Packets then move on to the router that connects to the desired memory controller. Routing on the way back down is just as simple, with one of the two possible middle level routers being chosen randomly. This routing simplicity also has latency advantages because the routing function takes less time in hardware.

3.1.3 Tapered Fat-Tree on TILE64

Unlike the original mesh network which has a very regular placement of routers and wires, place-and-route for a non-mesh network is fairly difficult. As a starting point, I use the router placements suggested in [4] for the TFT. Figure 3 illustrates the

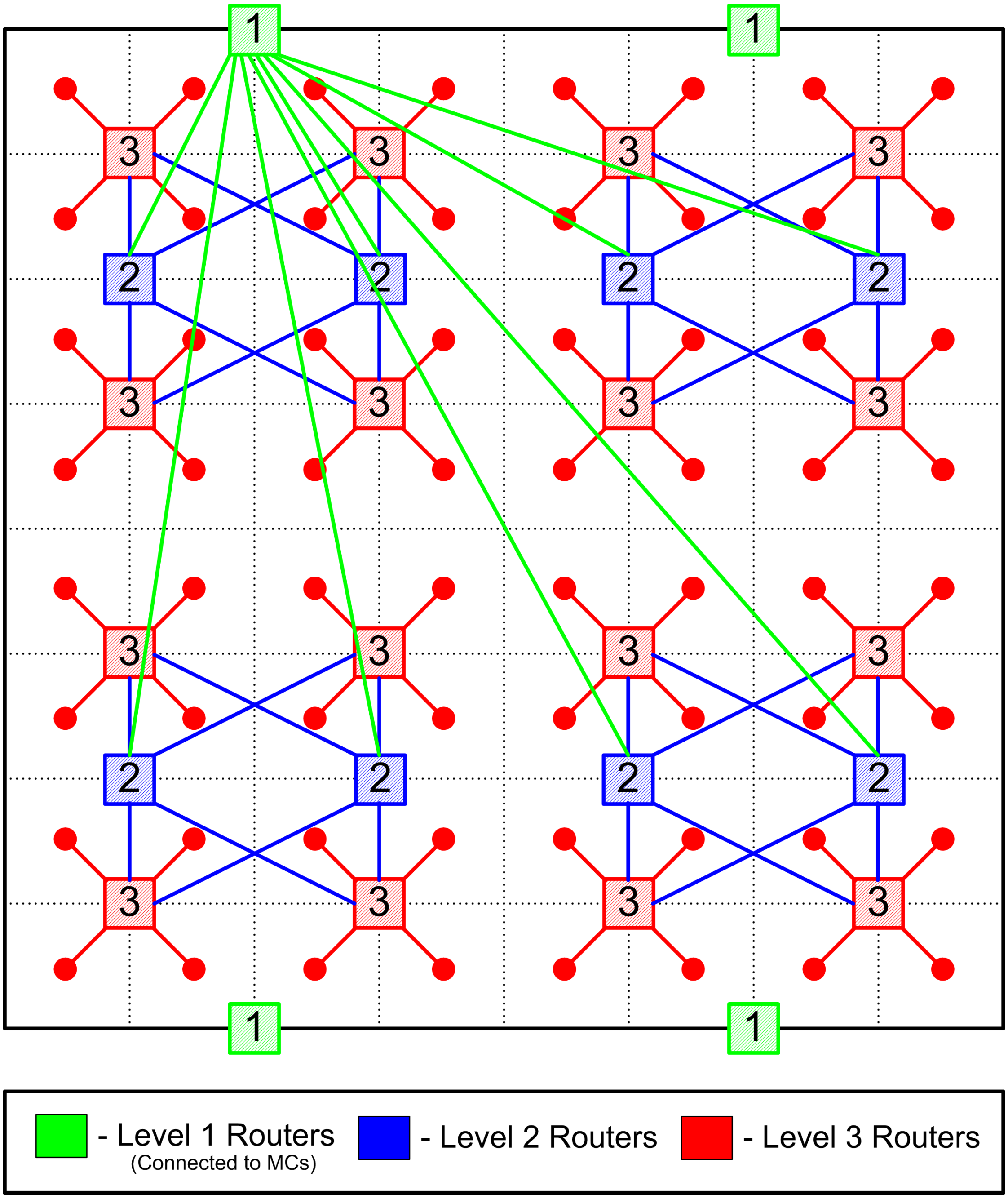

Figure 3: Tapered Fat-Tree mapped onto TILE64. The connections only represent logical connectivity. A possible physical layout is shown in Figure 5.

placement of the TFT routers and their logical connectivity mapped onto the TILE64. Due to the asymmetry, density, and lengths of the wire connections, the inter-router connections add significant power and delay. I discuss these issues in Section 4.

Chapter 4

Evaluation Models

4.1 Wire Model

[1]The propagation delay of a signal across a wire is proportional to $R_{wire}C_{wire}$. Because both R_{wire} and C_{wire} grow linearly with the length of the wire, the delay across a wire increases quadratically with wire length. By inserting a chain of repeaters (inverters) along the wire, designers can make the delay grow only linear with wire length. To achieve the minimal delay, the repeaters must be sized and placed along the wire such that the fanouts the repeaters drive form a geometric series.

Achieving minimal wire delay is a very difficult problem to solve, not only mathematically, but also in terms of physical design. For a given wire length, designers must first determine the number of repeaters that achieves the best tradeoff in terms of power, area, and delay. Then, they must size and place the repeaters such that the fanouts form a geometric series. Since most semiconductor companies use standard cells, the optimal buffer sizes are likely not part of the standard cell library, so the closest ones

---

[1] The work in this chapter was done in collaboration with Allen Lee for a CS258 class project. All mistakes are my own.

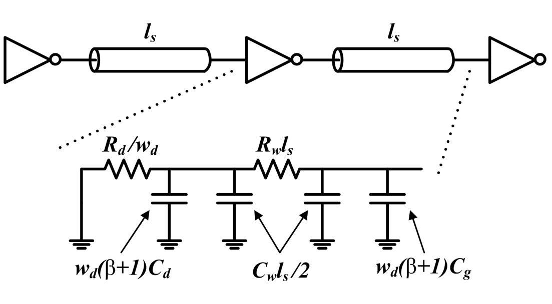

Figure 4: First-order repeater model

must be chosen. Even after choosing the standard cell inverters and determining the optimal placements, it may not be possible to place them at the ideal spots. Globally repeated wires require many via cuts from the upper layer down to the substrate, and the repeaters may be too large to fit into the substrate if the chip area is heavily utilized.

For these reasons, I use the first-order repeated wire delay model described in [4][18] and illustrated in Figure 4. Namely,

1. Repeaters are identically sized and evenly spaced.
2. The effects of inductance are not modeled.

I assume that each repeater has a PMOS/NMOS ratio of 2 and has twice the drive strength of a minimum-sized inverter.

The delay across a single repeater stage (T_s) is calculated as:

$$\begin{aligned} K_0 &= R_d(1+\beta)(C_d+C_g) \qquad (4.1)\\ K_1 &= \frac{R_d}{w_d}C_w + w_d R_w(1+\beta)C_g \qquad (4.2)\\ K_2 &= \frac{1}{2}R_w C_w \qquad (4.3)\\ T_s &= 0.69(K_0 + K_1 l_s + K_2 l_s^2) \qquad (4.4) \end{aligned}$$

Here, R_d is the equivalent resistance of the repeater, β is the PMOS/NMOS ratio, C_d is the diffusion capacitance, C_g is the gate capacitance, w_d is the width of the repeater, C_w is the capacitance per millimeter, R_w is the resistance per millimeter, and l_s is the length of the segment. The total wire delay across n stages (T_w) is:

$$T_w = nT_s \qquad (4.5)$$

4.1.1 Technology Model

Because the router power models are based on the 100nm process, I obtained and derived physical parameters for the wires corresponding to this technology node from the 2003 ITRS Roadmap [3]. I discuss the router power models in more detail in Section 4.2. Tables 2 and 3 present the relevant parameters.

4.1.2 Wire Layout

Figure 5 illustrates the physical layout of the wires for the routers at each level. I approximate the length of each wire as the Manhattan distance between the endpoints.

Table 2: Model Characteristics and Constants

Parameter	Symbol	Value	Units
Supply Voltage	V_{DD}	1.0	[V]
Clock Frequency	f	750	[MHz]
Tilera Tile Width	W_t	2.6025	[mm]
Gate Capacitance	C_g	1.95E-16	[F/μm]
Diffusion Capacitance	C_d	1.77E-15	[F/μm]
Resistivity of Copper	ρ_{Cu}	1.72E-8	[Ω-m]
Repeater Eq. Resistance	R_d	1138.4	[Ω-μm]
NMOS Leakage Current	$I_{off,N}$	10	[nA/μm]
PMOS Leakage Current	$I_{off,P}$	10	[nA/μm]
PMOS/NMOS Ratio	β	2	

Table 3: Interconnect Characteristics

Parameter	M1	Int.	Global	Units
Pitch	240	320	475	[nm]
Width	120	160	320	[nm]
Thickness	75	94	113	[nm]
A/R	1.6	1.7	2.1	
R_w	1.91E3	1.14E3	4.75E2	[Ω/mm]
C_w	2.00E-13	1.85E-13	1.31E-13	[F/mm]

I assume that Level 3 routers use low level metal wires, Level 2 routers use intermediate wires, Level 1 routers use global wires, and repeaters are placed every 0.5mm.

Because Tilera has not revealed the die size of the Tile64, I must estimate its dimensions. One source [22] postulated that the die size is approximately four times that of the quad-core Intel Core 2 Extreme QX6850, but this approximation results in such a large area that the yield would be virtually zero. Instead of using this estimate, I compare the transistor count of the Tile64 to the transistor count of NVIDIA's G80 chip, which was fabricated with the same process. The G80 GPU has 681 million transistors with a die size of 480mm^2 (21.9mm × 21.9mm) [1]. Because the Tilera has 615 million transistors [12], I approximate the Tile64 die area as

$$\frac{615\text{M transistors}}{681\text{M transistors}} * 480\text{mm}^2 = 433.48\text{mm}^2$$

This area corresponds to dimensions of 20.82mm × 20.82mm. At eight cores to a side, the width of one core (W_t) is thus 2.6mm. Due to the variability in size of transistors, this is a very rough estimate.

Table 4: Multi-Cycle Wire Delays ($t_{cycle} = 1.33$ ns)

Router Level	Paths	Length	n	T_w(ns)	Cycles
Level 1	P_1-P_4	W_t	5	1.22	1
Level 2	P_1-P_2	W_t	5	1.06	1
	P_3-P_4	$3 \cdot W_t$	15	3.18	3
Level 3	P_1-P_2	$3 \cdot W_t$	15	2.17	2
	P_3	$5 \cdot W_t$	26	3.61	3
	P_4-P_6	$7 \cdot W_t$	36	5.06	4
	P_7	$9 \cdot W_t$	46	6.50	5
	P_8	$11 \cdot W_t$	57	7.95	6

4.1.3 Multi-Cycle Wire Delay

As Figure 5 shows, the tapered fat-tree topology requires connectivity between points that are located at nearly opposite sides of the chip. Using the wire delay model described earlier in this section, I determined that these long paths require multiple cycles. Table 4 summarizes the results of my analysis.

4.2 Power Model

Because power is becoming the limiting constraint in modern multiprocessors, I model power dissipation in the network simulations to compare the energy efficiency of the different topologies. I consider the static and dynamic power dissipated both within the routers and in the inter-router interconnect. The intra-router energy measurements are based on the Orion energy models [24] using the 100nm technology node.

I assume a clock frequency of 750MHz and supply voltage of 1.0V, the same parameters used by the actual TILE64 [12].

Although the actual TILE64 multiprocessor uses the 90nm process, I was not able to obtain accurate power models for the routers at this technology node. However, because I am only interested in comparing the relative energy efficiency of the topologies, applying the same model to the different schemes is sufficient for comparison purposes.

4.2.1 Router Power

As a flit traverses a router, switching power ($P_{router,dyn}$) is consumed at the each of the functional units: the input buffer (P_{buf}), the matrix crossbar (P_{xbar}), and the arbiter (P_{arb}):

$$P_{router,dyn} = P_{buf} + P_{xbar} + P_{arb} \tag{4.6}$$

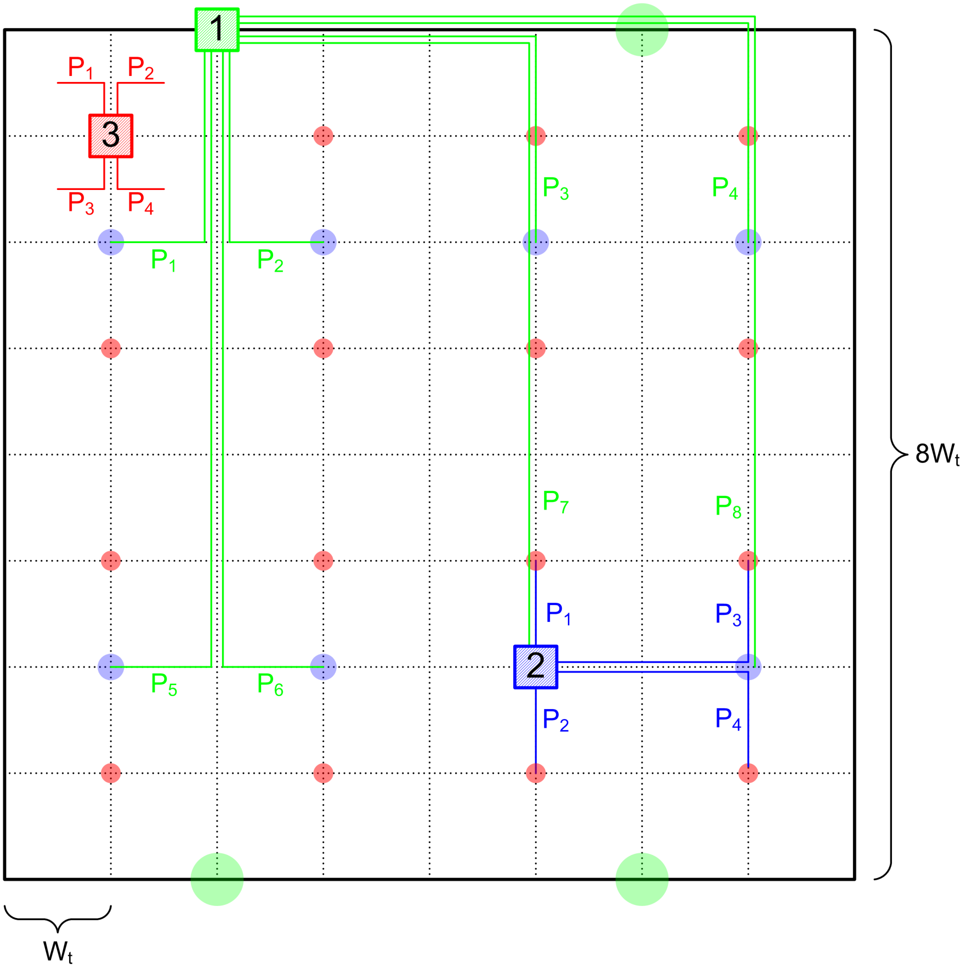

Figure 5: Physical layout of TFT on TILE64

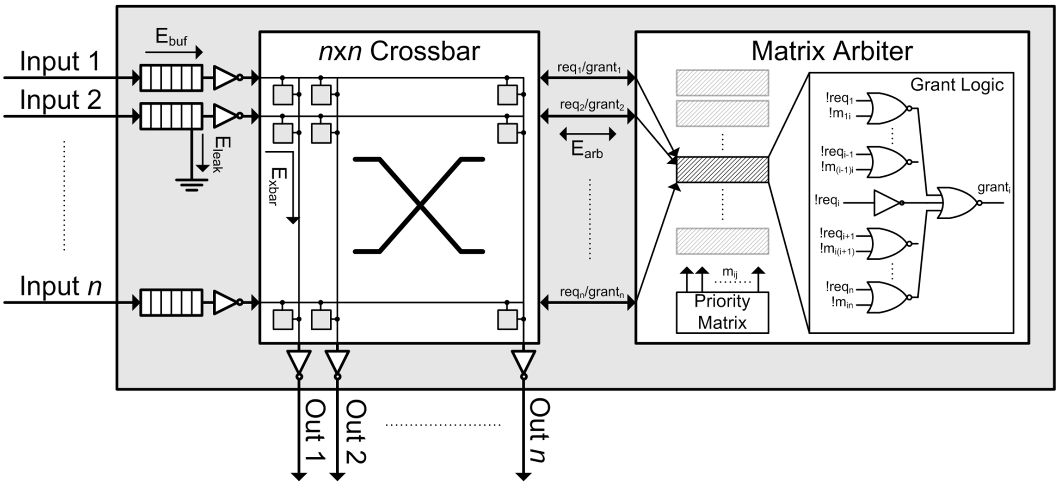

Figure 6: Input-buffered on-chip router with matrix arbiter.

Each functional unit is made up of a set of atomic components, i.e., elements consisting of one or more capacitors that have identical switching behavior. For an atomic component x with switching capacitance C_x and activity factor α, the average switching power $P_{x,sw}$ is calculated as:

$$P_{x,sw} = \frac{1}{2}\alpha C_x V_{DD}^2 f \tag{4.7}$$

The activity factor α is calculated individually for each atomic component of each functional unit from the load on the router. Here, load means the probability that a flit arrives at a single input buffer. To determine the appropriate load for each router, simulations keep track of the total flit activity at each of the routers.

As technology scales to deep submicron processes, subthreshold leakage power becomes a significant component of overall power dissipation. To account for leakage power in the simulations, I adopt the model detailed in [5]. The leakage power (P_{leak}) is calculated as:

$$P_{leak} = I_{leak} V_{DD} \tag{4.8}$$

Here, I_{leak} is the leakage current of the router buffers. Figure 6 illustrates the router I modeled with the energy components labeled. More details about the breakdown of the individual energy components can be found in [24]. The matrix arbiter used in the power model is different from the round-robin arbiter actually implemented in the simulation. I do not believe this would have any significant effect on the outcome of the simulation though, because both types of arbiters exhibit strong fairness, meaning both serve their requesters equally often dally:interconnection.

4.2.2 Interconnect Power

Although the Orion models provide accurate power models within the routers, they do not account for power dissipated in the inter-router interconnect. Because the TFT requires many global connections, interconnect power consumption may be significant. I calculate the static ($P_{w,stat}$) and dynamic ($P_{w,dyn}$) power dissipated driving a wire segment (l_s) based on the equations disclosed in [4]:

$$\begin{aligned} P_{w,dyn} &= \alpha(w_d(C_g + C_d) + C_w l_s) f V_{DD}^2 \\ P_{w,stat} &= \frac{1}{2} w_d(I_{off,N} + \beta I_{off,P}) V_{DD} \\ P_w &= P_{w,dyn} + P_{w,stat} \end{aligned}$$

Here, α is the activity factor on the wire obtained from the simulations.

Chapter 5

Simulation Environment

I wrote a detailed cycle-accurate flit-level simulator, consisting of 6000 lines of new code, implemented on top of an existing simulation framework developed by Christopher Batten of MIT. The simulator provides completion times and activity factors for every router and link. These activity factors were then used to provide power estimates. The Tile64 was modeled after the description in 2.1. The TFT networks used the wire delays listed in Table 4.

5.1 Description of Simulator Operation

The simulator consists of a number of nodes that are the sources, sinks, or memories, connected to the TDN and MDN networks. The sources and sinks represent the processors' L2 caches injecting cache requests onto the TDN and MDN. These packets enter the TDN network if they are a fill request, or the MDN if they are a writeback request (because I assume that all writebacks are going to off-chip memory, as you would not write to another core's L2). Packet sizes were as according to Table 6. Fill requests always travel to a directory cache that is on-chip to discover the location of the fill data. The network (MDN or TDN) over which a packet travels can be seen in Figure 8. Figure 8 shows the network path in the system where the MDN is a TFT,

Table 5: PARSEC Applications Simulation Parameters

Bencmark	Working Set Size	Sharing	Injection Rate	% Shared Access	% Fills
streamcluster	small	small	0.027	0.1	99
blackscholes	small	large	0.00014	19.0	33
canneal	large	small	0.023	0.1	66
x264	large	large	0.0041	50.9	68

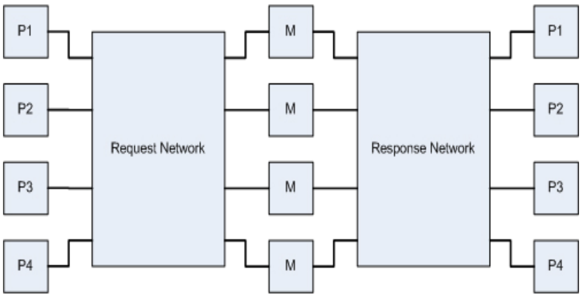

Figure 7: Two physically separate networks, for requests and replies

the only difference when the MDN is a mesh (the Tile64 system) is that fill responses from remote nodes are sent via the MDN instead of the TDN. The separate request and response networks connecting the source nodes to the memories can be visualized in Figure 7.

The packet is broken up into flits (all of which are 32 bits wide on both the MDN and TDN), which get buffered at the input buffer to the router that connects the source to the network. The router examines the destination field of the first flit to decide which output port to request arbitration for. The per-output arbiters use round-robin arbitration to decide which flit gets the grant for this cycle; a hold line is invoked until the tail flit of a packet passes to prevent flits from different packets from getting interleaved.

Once the router has perfomed switch allocation, the winning flits are then placed into a pipeline that simulates the intra-router delay. Flits that leave the router on this cycle are moved into the inter-router links, which delay their inputs based on the actual link length between the routers that it connects.

The flit makes its way through the network as described, until it arrives at the directory cache (inside a sink) or a memory, where the flits of a packet are reassembled. Directory caches transform fill queries into fill requests if the query is a hit at the directory cache, or they forward the query to an off-chip directory if it is a miss. Fill requests are forwarded to the appropriate destination, be it an on-chip remote node, or an off-chip memory. Fill requests received at an on-chip remote node are turned into fill responses and sent back to the original requesting source.

The memory changes requests into the appropriate response, placing it in a pipeline to delay the response for 52 cycles (the memory latency). The response then makes its way back to the source that originally issued the request. Queries that could not be satisfied at the on-chip directory cache produce a fill request that is either satisfied

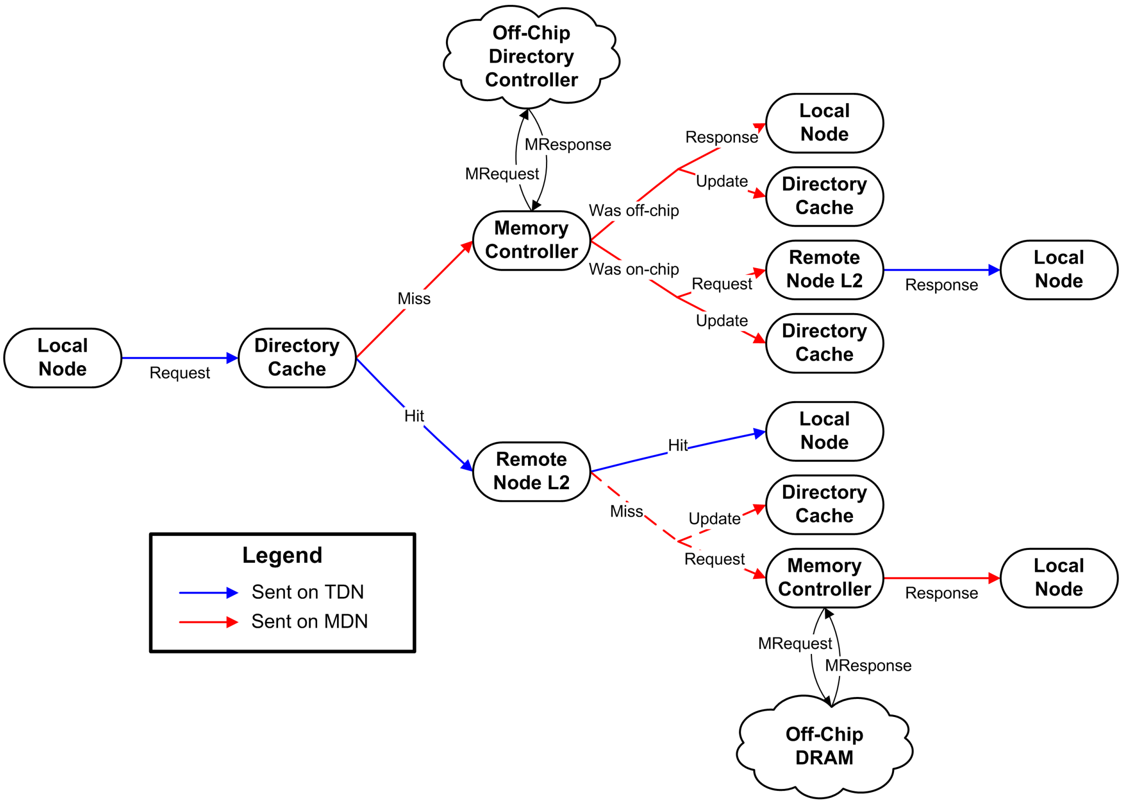

Figure 8: Protocol for TFT memory traffic

at the off-chip memory, or sent on-chip to the remote node holding the data.

5.2 Benchmark Description

I performed synthetic benchmarks, whose parameters were based on a selected set of the PARSEC benchmarks. I chose four that would provide a full range of small and large working sets, and low and high amounts of sharing. This would give a full range of stress on the L2DN, as well as the interprocessor interconnection network, and also capture interactions between the two. See 5 for the full simulation parameters. Per

Table 6: Simulation Packet Sizes

Packet Type	Size (bits)
Directory Fill Query	32
Fill Request	32
Fill Response	288
Store Request	288
Store Response	32

processor injection rate is the probability that a given processor would inject a request into the network on a cycle. Percent shared accesses refers to what percent of requests must go off-chip to be satisfied. Fills are the probability that a request is a cache line fill (generated by a load or allocate on store), otherwise it is a writeback. I assumed that any shared requests could be satisfied by a node on-chip. These numbers were calculated by taking the off-chip memory traffic rate from the PARSEC benchmark, dividing by the percent of unshared accesses to calculate total cache traffic. The relative ratios of fills and writebacks were used to determine what type of traffic to generate. Note that I did not implement a cache coherence protocol, or run the PARSEC benchmarks. I used the characterizations of the applications' traffic to motivate my synthetic benchmarks.

To simulate a real processor, each source is limited to a maximum of eight outstanding requests at any given time. This number was chosen by examining the number of outstanding requests that allowed each network to finish an average workload in minimum time, while stalling sources that were not immediately able to inject their traffic into the network.

At each cycle, the source uses the injection rate probability to decide whether to generate a request. If it generates a request but it already has eight outstanding requests, or if there is already another request by this source attempting to enter the network, the source is stalled and does not do any 'work' during this cycle. Otherwise, the source completes a unit of work each cycle. Benchmarks are completed when a certain amount of work is done by the processors. Clearly, benchmarks will complete faster if sources do not spend too much time stalling.

Chapter 6

Results

6.1 Benchmark Completion Time

To compare the relative performance of the different topologies, I chose a workload for each benchmark and executed the same workload on each of the topologies. I normalize the completion time of each benchmark to the time it takes to finish executing when the MDN is comprised of the original mesh topology. The results are shown in Figure 9. It should be noted that when I refer to mesh or tree topology, or mesh or tree network, I am referring to the full simulation setup with a mesh TDN (always), and either a mesh MDN or a tapered fat tree TDN. All energy, power, and completion time numbers include the activity of both the TDN and MDN. This means the relative power savings of the TFT to the mesh is actually greater than that shown here, but because the TDN is also part of the system I am simulating, I thought it a fairer comparision if I included the TDN's energy consumption.

The simulations indicate that for programs with large amounts of sharing, such as `blackscholes` and `x264`, the TFT offers little advantage over the original Mesh network. These applications require only a small amount off-chip bandwidth, so most of the memory traffic stays on the TDN, which is common to both networks I tested. The fact that the TFT network has a slightly faster completion time is due to packets spending less time in transit, as shown in Figure 10. As expected, the higher traffic injection rate of `x264` results in a small amount of queueing, of which there is almost none in `blackscholes`.

For applications with small amounts of sharing, such as `streamclrefuster` and `canneal`, most of the memory traffic goes off-chip. As demonstrated in Figure 11, the packets are spending the majority of their time in queues, waiting to make forward

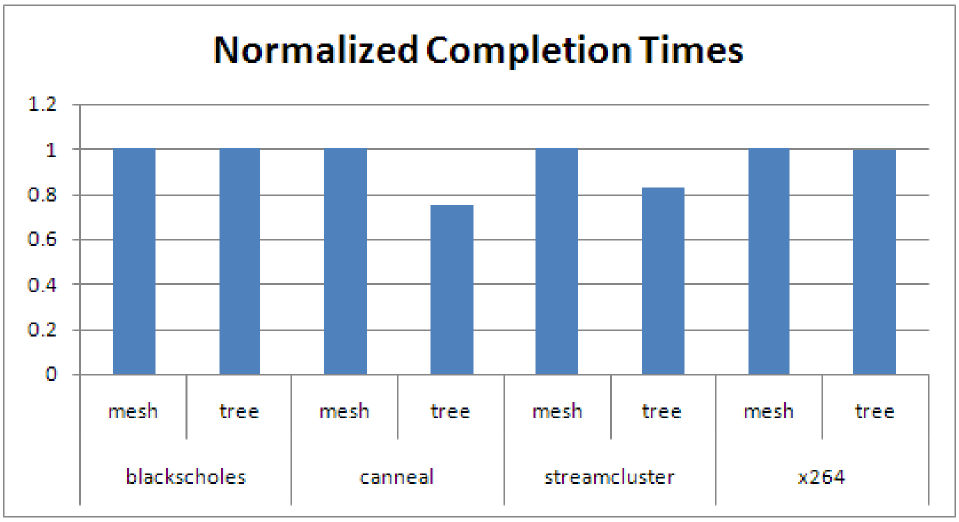

Figure 9: Completion times of benchmarks normalized to the completion time with the Mesh topology.

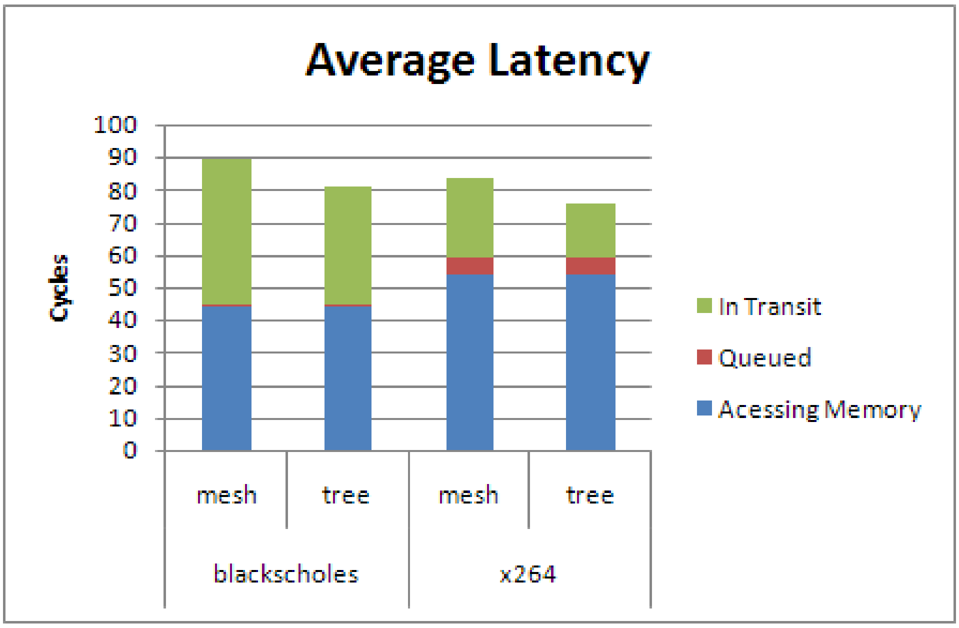

Figure 10: Average latency of packets for the blackscholes and x264 benchmarks.

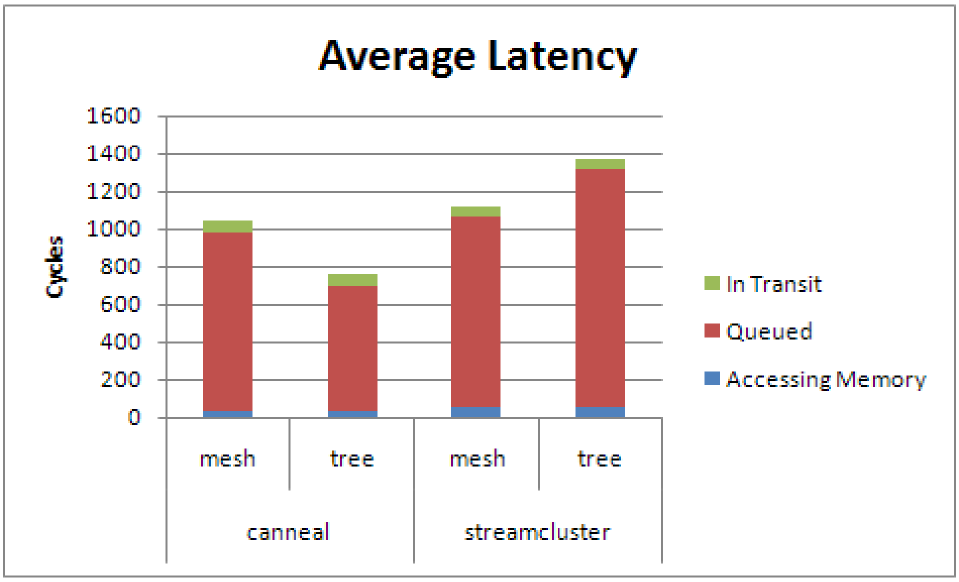

Figure 11: Average latency of packets for the canneal and streamcluster benchmarks.

Table 7: Throughput of Selected MDN Routers

Benchmark	Router	Topology	Throughput (bits/cycle)
streamcluster	memory controller, outgoing	mesh	3.3
	memory controller, incoming		29.0
	top level, outgoing	tree	4.0
	top level, incoming		35.0
canneal	memory controller, outgoing	mesh	14.0
	memory controller, incoming		27.0
	top level, outgoing	tree	18.7
	top level, incoming		36.2

progress. At this point, the primary reason that the tree network completes faster is not due to the lower latency of the tree network, it is due to the increased throughput, shown in Table 7. In these benchmarks where congestion on the MDN becomes an issue, the TFT topology speeds up the completion time by 33.6% for `canneal` and 20.6% for `streamcluster`.

6.2 Power

The TFT topology shows a power savings for all of the benchmarks, as shown in Figure 12. Interestingly, a majority of the power dissipation appears to be from static leakage power because the routers on the MDN are fairly under-utilized. Even for memory-intensive applications, the average load for routers in the Mesh network

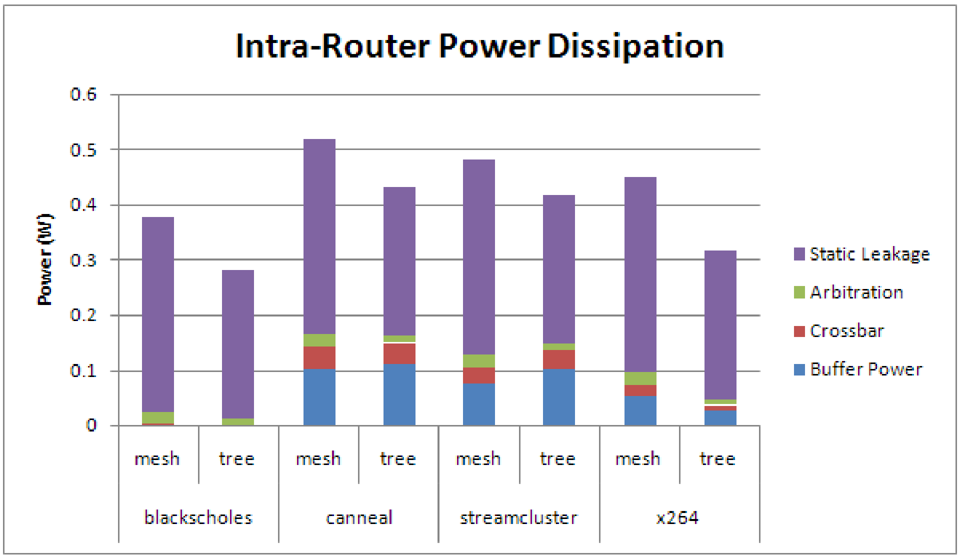

Figure 12: Power dissipated within the routers of the network.

remain below 0.09. In the TFT network, the load for the routers was below 0.08 for the routers going toward the memory controller. I observed larger activity factors for the TFT routers returning from the memory controller (as much as 0.565 for the Level 1 routers in the `canneal` benchmark), but these routers have a small number of input ports, so their dynamic power dissipation remains low.

These power measurements suggest that a majority of the power savings comes from reducing static power dissipation. Because the original Mesh network used 64 routers in the MDN, and the TFT topology uses only 56 routers with fewer input ports on average, the TFT appears to be a more energy-efficient topology.

The effects of the inter-router interconnect can be seen in Figure 13. The tree network dissipates less power than the mesh network for all benchmarks except for `canneal`. The reason for this is due to the higher operating throughput of the tree network than the mesh network, especially for this specific workload. This higher throughput is achieved due to the nature of the traffic distribution for the `canneal`: 33% are stores, compared to 1% stores for `streamcluster`. In `streamcluster`, the large number of loads makes a bottleneck at the memories on the return path. The 288-bit load responses that are returning on the MDN completely congest that part of the network. `canneal`, on the other hand, balances traffic types better. As store requests are nine times larger than load requests on the request network, and load responses are nine times larger than store responses, traffic is distributed more evenly among the request and response networks, allowing higher network utilization.

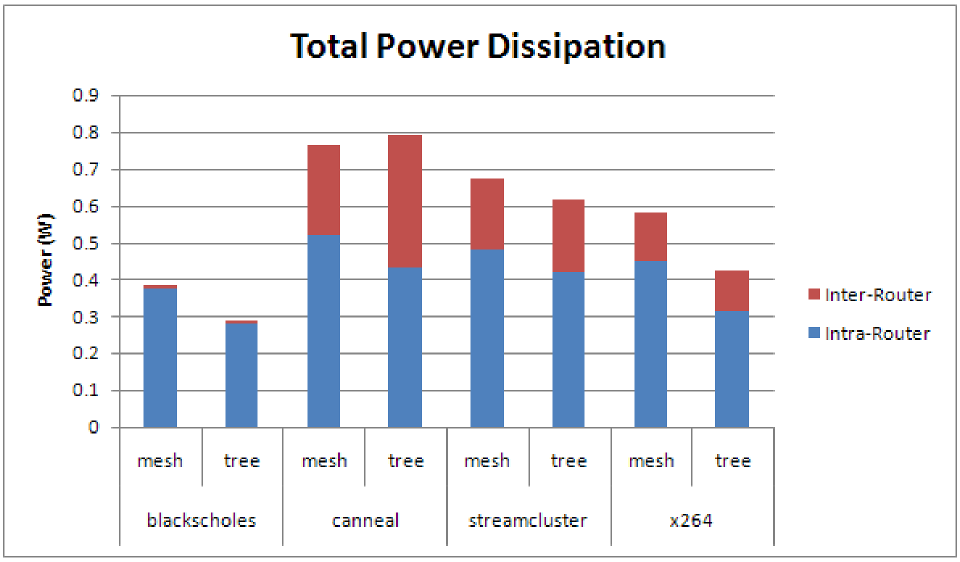

Figure 13: Total power dissipated in routers and interconnect.

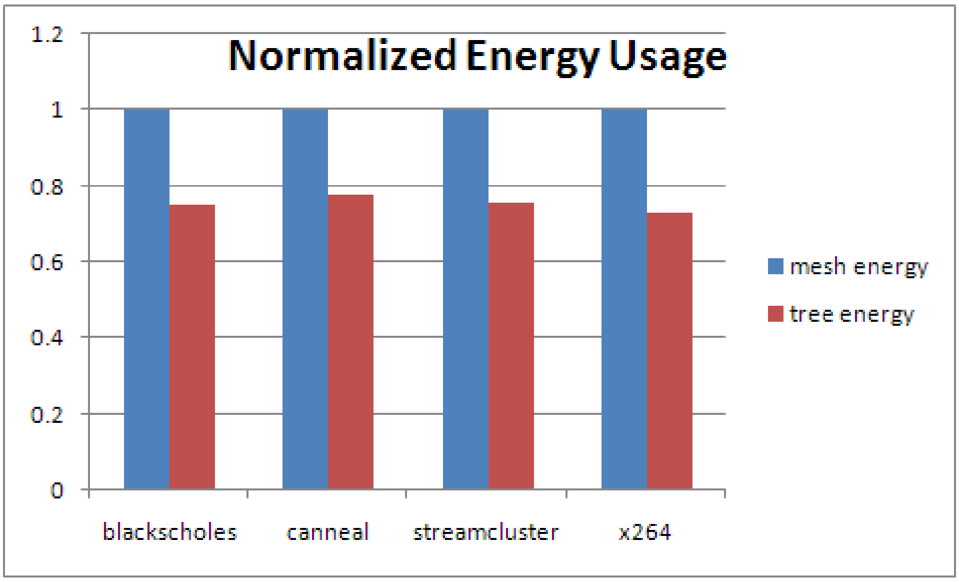

Figure 14: Relative energy dissipated in routers and interconnect.

This higher power usage does not cause an increase in energy required to complete the workload. To the contrary, the tree network uses less energy to finish all of the benchmarks. The greater power usage in `canneal` was due solely to the fact that the network was operating at a sufficiently higher throughput. It did more work per cycle, enough to result in higher power requirements despite lower per unit of work energy required. The total energy used to complete each workload is shown in Figure 14, which includes both intra-route energy and inter-router energy. The tree network consistently uses between 70%-80% of the energy consumed by the mesh network.

Chapter 7

Related Work

Interconnection networks have been studied for decades now, used in applications as diverse as a uniprocessor's out-of-order scheduling to supercomputers composed of hundreds of nodes. An excellent reference for all aspects of interconnection networks is [10]. Interconnection networks designed specifically for parallel architectures are given a thorough treatment in [6]. L2 to off-chip memory interconnects for CMPs have not been specifically researched before due to the relatively recent introduction of CMPs. Research into interconnection networks has largely concentrated on 'many to many' networks, as opposed to the 'many to few' design discussed in this thesis.

The mesh networks used by the Tile64 are instances of k-ary n-cubes, which have been popular interconnection networks for years. Hypercubes were used for many parallel computers before Dally pointed out the benefits of lower radix k-ary n-cubes when under realistic constraints such as wiring and pin density [8]. Low dimensional meshes have been very popular interconnects since. Fat trees were first introduced in [20]. Generalized fat trees, of which the tapered fat tree used in the thesis is a variant of, were described in detail in [23]. The tapered variant that I employ was analyzed in terms of power, performance, and area usage in [4] and performed well, prompting me to examine it for a special-purpose L2 to memory interconnect.

Virtual channel flow control, employed in the modified design to allow the TDN to run without deadlock, was introduced in [7] and further developed [9]. It was used extensively in [21] to allow packets to route adaptively over k-ary n-cubes. Virtual channel flow control has been used in many supercomputer networks, such as the Cray T3D [19] and SGI Spider[17].

The power modeling of routers is a topic of importance. In the context of on-chip networks, routers microarchitecture is covered in [25]. The Orion power model used in this thesis was developed based on examining two commercial routers, the Alpha 21364 and InfiniBand routers [26].

Many different benchmarks for parallel workloads have been proposed over the years, the most commonly used (until recently) were the SPLASH and SPLASH2 benchmarks [2]. The authors of the PARSEC benchmark suite believe that the SPLASH2 benchmarks are now outdated and not representative of future CMP workloads [13]. The increased demands of the PARSEC benchmarks will be a primary motivating factor for re-examining the off-chip memory interconnect.

Chapter 8

Conclusion

In this thesis, I proposed a detailed solution for providing an energy-efficient topology for off-chip processor-memory interconnect to improve the performance of memory-intensive parallel applications. I have demonstrated that the tapered fat-tree topology improves performance by up to 33.6% for applications with high off-chip bandwidth requirements without reducing the performance of those with mostly on-chip memory traffic. This is primarily due to the increased throughput made possible by the TFT topology, and secondarily because of the reduced latency. By reducing the number of total number of routers and simplifying the routers' crossbars, I have also shown that the TFT offers substantial energy savings in all of the benchmarks, regardless of working set size or the amount of cache-line sharing. This energy savings translates into lower power requirements for all of the benchmarks but the most intensive.

As the number of cores on a single chip continues to grow and the number of memory access points scales less aggressively, I feel the study of similar 'many to few' topologies warrants additional attention. Although design and physical layout of non-mesh networks is more difficult, and possibly requires more wires, the potential power savings and performance improvements may soon outweigh the added design complexity.

Acknowledgments

This thesis is dedicated to my grandfather, who was a role model to me, and always extremely supportive of my education as well. I would not have had the opportunities that I have had without him.

I would like to thank Allen Lee for his excellent work on the power models appearing in this thesis, without his help I would have been lost. Professor Krste Asanovic and Professor John Kubiatowicz granted me their insight and had invaluable ideas, and their patience is greatly appreciated. I would especially like to thank Christopher Batten, whose brainstorming was the basis for this thesis. His simulation framework made all the simulation possible, and his encouragement provided a light at the end of the tunnel. I would also like to thank David Wentzlaff for providing details about the Tile64 that were intrsumental to my analysis.

Bibliography

[1] GeForce 8 Series. http://en.wikipedia.org/wiki/GeForce-8-Series.

[2] Stanford Parallel Applications for Shared Memory. http://www-flash.stanford.edu/SPLASH/.

[3] *International Technology Roadmap for Semiconductors (ITRS)*, 2003.

[4] James Balfour and William J. Dally. Design Tradeoffs for Tiled CMP On-Chip Networks. In *International Conference on Supercomputing (ICS)*, 2006.

[5] Xuning Chen and Li-Shiuan Peh. Leakage Power Modeling and Optimization in Interconnection Networks. In *International Symposium on Low Power Electronics and Design (ISLPED)*, August 2003.

[6] David E. Culler and Jaswinder Pal Singh. *Parallel Computer Architecture*. Morgan Kaufmann, 1999.

[7] William J. Dally. Virtual channel flow control. In *International Symposium on Computer Architecture (ISCA)*, May 1990.

[8] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. In *IEEE Transactions on Computers*, June 1991.

[9] William J. Dally. Virtual-channel flow control. In *IEEE Transactions on Parallel and Distributed Systems*, March 1992.

[10] William James Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[11] Asanovic et al. The Landscape of Parallel Computing Research: A View from Berkeley. Technical report, UC Berkeley, Dec 2006.

[12] Bell et al. TILE64 Processor: A 64-Core SoC with Mesh Interconnect. In *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2008.

[13] Bienia et al. The PARSEC Benchmark Suite: Characterization and Architectural Implications. Technical report, Princeton University, Jan 2008.

[14] Konstadinidis et al. Implementation of a Third Generation 16-Core 32-Thread Chip-Multithreading SPARC Processor. In *International Solid-State Circuits Conference (ISSCC)*, Feb 2008.

[15] Leon et al. A Power-Efficient High-Throughput 32-Thread SPARC Processor. In *International Solid-State Circuits Conference (ISSCC)*, Feb 2006.

[16] Wentzlaff et al. On-Chip Interconnection Architecture of the Tile Processor. In *Micro, IEEE*, 2007.

[17] Mike Galles. Scalable pipelined interconnect for distribued endpoint routing: The SGI SPIDER chip. In *Symposium on Hot Interconnects*, 1996.

[18] Ron Ho, Kenneth W. Mai, and Mark A. Horowitz. The Future of Wires. In *Proceedings of the IEEE*, April 2001.

[19] R. E. Kessler and J. L. Schwarzmeier. Cray T3D: a new dimension for Cray Research. In *IEEE Computer Society International Conference (COMPCON)*, Feb 1993.

[20] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. In *IEEE Transactions on Computers*, Oct 1985.

[21] Daniel H. Linder and Jim C. Harden. An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes. In *IEEE Transactions on Computers*, Jan 1991.

[22] Guy Macon. TILE64 Embedded Multicore Processors. https://www.gelato.unsw.edu.au/archives/comp-arch/2007-October/013123.html.

[23] Sabine Ohring, Maximilian Ibel, and Sajal Das. On Generalized Fat Trees. In *Parallel Processing Symposium*, 1995.

[24] Hangsheng Wang. A Detailed Architectural-Level Power Model for Router Buffers, Crossbars and Arbiters. Technical report, Princeton University, Jan 2004.

[25] Hangsheng Wang, Li-Shiuan Peh, and Sharad Malik. Power-driven Design of Router Microarchitectures in On-chip Networks. In *Proceedings of MICRO 36*, Nov 2003.

[26] Hangsheng Wang, Li-Shiuan Peh, and Sharad Malik. A power model for routers: Modeling Alpha 21364 and InfiniBand routers. In *MICRO 23, IEEE*, 2003.