HIGH-RADIX INTERCONNECTION NETWORKS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL
ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

John Kim
March 2008

© Copyright by John Kim 2008
All Rights Reserved

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(William J. Dally) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Mark Horowitz)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Kunle Olukotun)

Approved for the University Committee on Graduate Studies.

Abstract

Over the past twenty years, the pin bandwidth available to a chip has increased by approximately an order of magnitude every five years – a rate very similar to Moore's Law. This increasing pin bandwidth can be effectively utilized by creating *high*-radix routers with large number of skinny ports instead of low-radix routers with fat ports. The use of high-radix routers leads to lower cost and better performance, but high-radix networks present many difficult challenges. This thesis explores challenges in scaling to high-radix routers, including topology, routing, and router microarchitecture.

Topology is a critical aspect of any interconnection network as it sets performance bounds and determines the cost of the network. This thesis presents a cost-efficient high-radix topology referred to as the flattened butterfly topology which exploits the availability of high-radix routers. Compared to a folded-Clos topology, the flattened butterfly provides approximately 2× reduction in cost per performance on balanced traffic while maintaining the same cost per performance on adversarial traffic pattern. Given the topology, routing determines the path between the source and its destination. Proper routing is required to exploit the path diversity available in a high-radix network and we discuss the advantages of using adaptive routing in high-radix networks as well as how non-minimal routing is critical to properly exploiting the flattened butterfly topology.

Conventional microarchitectures do not scale to high radix since the complexity of the allocators in the routers scale quadratically with the radix. This thesis presents a hierarchical router organization that results in a distributed, complexity-effective microarchitecture and maintains high performance. As a case study, the implementation of this microarchitecture in the Cray YARC router with 64 ports will also be presented.

With the recent increase in the number of cores on a single-chip, the on-chip interconnection network that connects the on-chip cores and memory together will become more critical in determining the overall performance and cost of the entire chip. In the last part of this thesis, we expand the use of high-radix routers to on-chip networks and show how high-radix routers and the flattened butterfly topology can be mapped to on-chip networks.

Acknowledgements

I am truly grateful as I finish this thesis. This thesis would not have been possible without the support of many people. I feel I have put in a lot of effort but it is truly through the grace of God that I write this acknowledgement section as I wrap up my thesis. I thank God for the opportunity to undertake this work and the opportunity to meet and come across the following people who made this thesis possible.

Graduate school at Stanford would not have been possible without the support of my parents to get me here so I would like first thank my parents for their support. Without their dedication and sacrifice, none of this would have been possible.

I am deeply indebted to my advisor Professor Bill Dally. I have been very fortunate to be able to have worked under his guidance. His expertise and breadth of knowledge provided great insights into research and made this thesis possible. I would like to thank Professor Mark Horowitz and Professor Kunle Olukotun for serving on my committee as well as providing feedback on the thesis. I would also like to thank them for the different opportunities during the early days of my graduate school as I had the opportunity to TA for Professor Olukotun for couple of quarters and had the opportunity to work on the Smart Memories project under Professor Horowitz. I would like to thank Professor Andrea Montanari for serving as the chair of my oral committee.

I would like to thank Dr. Steve Scott and Dr. Dennis Abts of Cray for their support in this research – especially Dennis with the constant e-mail exchanges to understand the different issues and exchange ideas.

I would like to thank the current and the former members of the Concurrent VLSI Architecture (CVA) group, for their help and support throughout these years.

I especially thank the *interconnecters* (Brian, Ajrun, and Amit) and enjoyed the numerous network subgroup meetings and the stimulating discussions. Many thanks to Brian for his help in getting my first paper published. I would also like to acknowledge Patrick, Amit, and Tim for being extremely supportive and understanding office mates. I also thank James and David for proof reading parts of this thesis. I thank Ji-young for his help in finalizing this thesis and allowing me to remotely submit the thesis.

On a personal level, I would like to thank Pastor Park Jongwhan and his family for their support and prayers during my years at Stanford. I am grateful for their friendship. I would like to thank Pastor Seol and the members of Cornerstone Community Church for their support as well. I would also like to thank my family and friends for their support, especially my parents-in-law for encouraging me to finish my thesis in a timely and manner. And last but not least, to my wife Sun for her love, patience, and support as I finished my thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction to Interconnection Networks

An interconnection network can be defined as a "programmable system that enables fast data communication between components of a digital system" [22]. Any time two or more components of a digital system are connected, some form of an interconnection network is needed. An interconnection network involves sharing communication resources among multiple terminals and provides a structured way to organize communication among the different terminals.

Many different implementations of interconnection networks exist. A bus provides a very simple interconnection network in which all terminals communicate through a common bus as shown in Figure 1.1(a). However, as the number of terminals increases, the bus architecture does not scale and it also becomes difficult to provide high bandwidth. The crossbar is another implementation of an interconnection network that is widely used in which all the terminals are connected to each other through a single switch (Figure 1.1(b)). The crossbar provides point-to-point connections but the complexity of a crossbar is approximately $O(N^2)$ where N is the number of terminals connected to the crossbar.

To address these problems, multi-stage networks are often employed in interconnection networks as N increases. In a multi-stage network, the terminals are connected to *routers* that are in turn connected to other routers in the network. If there is

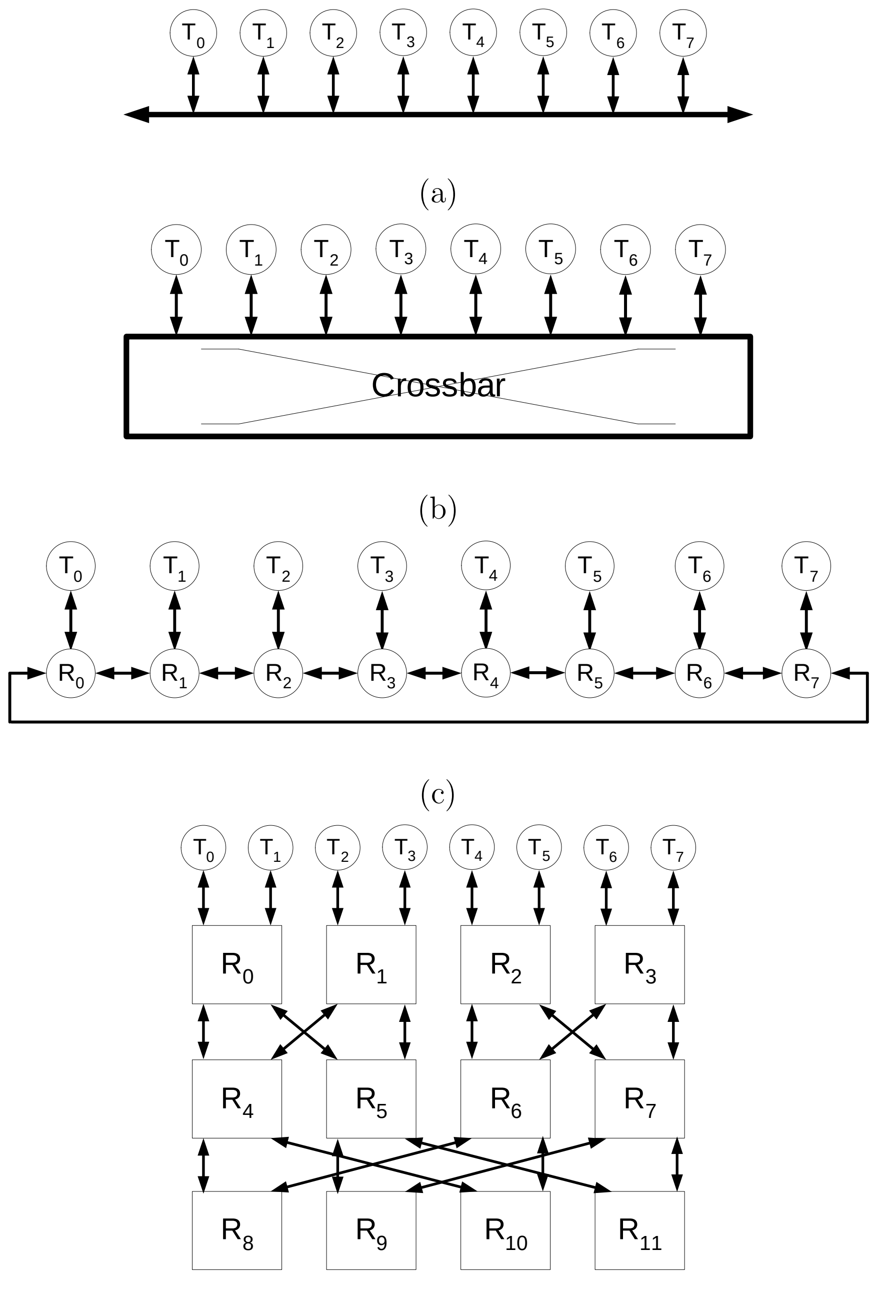

T_0 T_1 T_2 T_3 T_4 T_5 T_6 T_7
(a)
T_0 T_1 T_2 T_3 T_4 T_5 T_6 T_7
Crossbar
(b)
T_0 T_1 T_2 T_3 T_4 T_5 T_6 T_7
R_0 R_1 R_2 R_3 R_4 R_5 R_6 R_7
(c)
T_0 T_1 T_2 T_3 T_4 T_5 T_6 T_7
R_0 R_1 R_2 R_3
R_4 R_5 R_6 R_7
R_8 R_9 R_{10} R_{11}
(d)

Figure 1.1: Different implementations of an interconnection network. An 8-node network realized using (a) a bus, (b) a crossbar switch, (c) a ring, and (d) a fat-tree (folded-Clos) network.

a one-to-one correspondence between terminals and routers, a *direct* network such as the ring topology can be created(Figure 1.1(c)). If there are intermediate routers, the resulting multi-stage network is an *indirect* network such as the folded-Clos topology shown in Figure 1.1(d). However, unlike a bus or a crossbar, multi-stage networks do not provide a direct connection between the terminals and thus require terminals to send messages through one or more routers. As indicated from the four different examples shown in Figure 1.1, different interconnection networks offer different advantages and the optimal choice depends on the system requirements.

Regardless of the interconnection network being used, there are four common aspects that must be considered in the design of any interconnection network. The first three aspects deal with high-level network issues of an interconnection network.

- *Topology* - Topology is the roadmap of the network and defines how the channels and routers are connected. Topology is critical since it sets performance bounds for the network by determining the diameter of the network as well as the bisection bandwidth.

- *Routing* - Given the topology of an interconnection network, routing determines which path a packet takes from the source to its destination. An efficient routing algorithm is critical for approaching the performance bounds of the topology and load-balance the topology on adversarial traffic patterns.

- *Flow Control* - The flow control policy allocates the different resources in the network such as the buffers and the channels as the packet progresses from the source to its destination. Proper flow control is also required to prevent deadlock and livelock in the network.

Another important aspect of an interconnection network is the microarchitecture of the routers, which form the building block of an interconnection network.

- *Microarchitecture* - The microarchitecture defines how the router is organized, and includes the buffer and switch organization as well as the allocators used.

These four aspects of an interconnection network determine how the interconnection network is implemented and impacts the performance and cost of the network. In this thesis, we describe the use of *high*-radix routers within an interconnection network and investigates topology, routing and router microarchitecture aspects of high-radix networks. We next examine different applications which use interconnection networks and the importance of an interconnection network within a system is also discussed.

1.2 Importance of Interconnection Networks

Interconnection networks are found across a wide range of applications – they are widely used to connect processors and memories in multiprocessors [67, 68], as switching fabrics for high-end routers and switches [20], for connecting I/O devices [65], and for on-chip networks [25]. Across these different applications, the performance of most digital systems today is limited by their communication, not by their logic or memory.

Processor and memory technology have advanced according to Moore's Law – with the number of available transistors growing exponentially over time. This scaling has made interconnection networks more critical as wire density has scaled at a slower rate and wire delay has remained constant over time. Hence, the network has become a major factor in determining the overall performance and cost of the system

In a multiprocessor computer system, as processor and memory performance continues to increase, the performance of the interconnection network plays a central role in determining the overall performance, as well as the cost of the system. The latency and bandwidth of the network largely establish the remote memory access latency and bandwidth. This is also true for on-chip networks. As the number of components being integrated on chip increases, the on-chip network that connects these components will be critical for determining the performance benefits of future multicore processors.

In addition to performance, the cost of an interconnection network can account for as much as 1/3 of the overall system cost in large systems [61]. Thus, reducing the cost of an interconnection network is crucial to the overall system cost. Similarly,

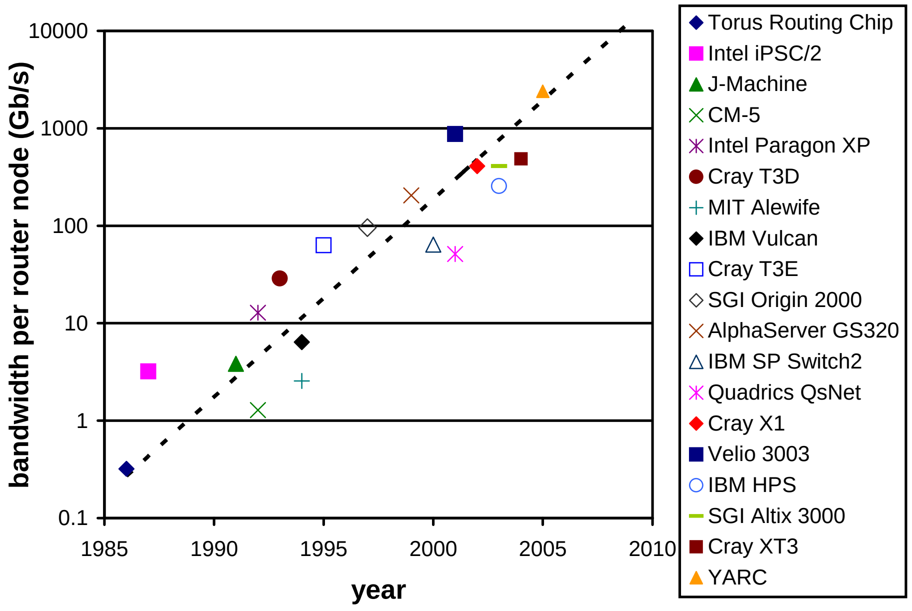


Figure 1.2: Router Scaling Relationship [24, 60, 27, 67, 4, 18, 53, 50, 29, 64, 71, 35]. The dotted line is a curve fit to all of the data.

the area and power concerns of future multicore processors needs to be addressed as the number of cores increases. In this thesis, we focus on how technology impacts interconnection networks and how the technology trend can be exploited to reduce the cost as well as the latency of the interconnection network and result in a more *efficient* system.

1.3 Technology Trend

Computer architecture is greatly impacted by technology as most notably illustrated by the impact of Moore's Law. As technology evolves, its impact on computer architecture needs to be properly re-evaluated. A technology that has great impact on interconnection networks is signaling technology and the resulting total pin bandwidth. Advances in signaling technology have enabled new type of interconnection networks based on *high*-radix routers which will be described in this thesis.

The trend of increase in pin bandwidth to a router chip is shown in Figure 1.2

which plots the bandwidth per router node versus time. Over the past 20 years, there has been an order of magnitude increase in the off-chip bandwidth approximately every five years – a rate very similar to Moore's Law. This increase in bandwidth results from both the high-speed signaling technology [34, 51] as well as the increase in the number of signals available to a router chip. The advances in technology make it possible to build single chips with 1Tb/s of I/O bandwidth today [32, 68], and by 2010, we expect to be able to put 20Tb/s of I/O bandwidth on a chip.

Most implementations have taken advantage of increasing off-chip bandwidth by increasing the bandwidth per port rather than increasing the number of ports on the chip. However as off-chip bandwidth continues to increase, this thesis will show it is more efficient to exploit this bandwidth by increasing the number of ports — building high-radix routers with *thin* channels — than by making the ports wider — building low-radix routers with *fat* channels. We show that using high-radix routers reduces hop count and leads to a lower latency and a lower cost interconnection network solution.

1.4 Contributions

This thesis includes several key contributions to the design of high-radix interconnection networks. We first focus on off-chip interconnection networks utilizing high-radix routers.

- *Motivation for High-Radix Routers* [45]. We describe how high-radix routers can exploit the increasing pin bandwidth and reduce the diameter of the network – resulting in both lower latency and lower cost.

- *Flattened Butterfly Topology* [43]. We propose a cost-efficient topology that can exploit high-radix routers. The flattened butterfly approaches the cost of a conventional butterfly network while providing similar performance/cost compared to a folded-Clos on adversarial traffic patterns. On benign traffic patterns, the flattened butterfly topology can provide 2× increase in performance/cost compared to a folded-Clos.

- *Adaptive Routing in High-Radix Networks* [42, 43]. High-radix networks present different challenges in adaptive routing. We show how transient imbalance can occur in a high-radix router with concentration and show how adaptive routing, if done properly, provides benefits over oblivious routing in a high-radix folded-Clos network. We also show how global adaptive routing is essential to fully realize the benefits of the flattened butterfly topology.

- *Hierarchical Router Microarchitecture* [45, 68]. Conventional router microarchitecture does not scale as the radix increases. We describe a router microarchitecture that can scale efficiently to high radix with minimal loss in performance. Because of its tiled organization, the microarchitecture is a complexity-effective design and its implementation in the Cray YARC router is also described.

From off-chip interconnection networks, we extend the use of high-radix routers to on-chip networks.

- *Flattened Butterfly Topology for On-Chip Networks* [40, 41]. Although on-chip networks provide very different constraints compared to off-chip networks, the use of high-radix routers in on-chip networks can provide similar benefits – including lower latency and lower cost. We propose using the flattened butterfly topology for on-chip networks and illustrate how it can be mapped as well as the router microarchitecture.

1.5 Outline

For the remainder of this thesis, we focus on the different aspects of high-radix interconnection networks. Chapter 2 discusses how the increasing pin bandwidth can be efficiently utilized by the use of *high*-radix routers. Chapter 3 proposes a cost-efficient topology referred to as flattened butterfly topology which is more cost-efficient than alternative topologies. Adaptive routing on high-radix networks is discussed in Chapter 4. A hierarchical router microarchitecture that can scale to high radix is presented in Chapter 5. In Chapter 6, we extend the work on high-radix interconnection networks to on-chip networks and show how the flattened butterfly can provide a more efficient on-chip network. Conclusions and future work are presented in Chapter 7.

Chapter 2

Motivation for High-Radix Routers

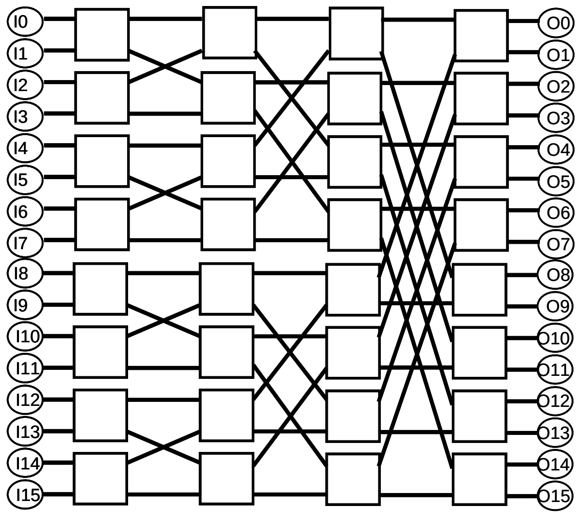
In this chapter, we describe the benefits of using high-radix routers and how they can exploit the increasing pin bandwidth that was shown earlier in Section 1.3. Using a simple conventional butterfly topology example, we illustrate the advantages of using high-radix routers and how this reduces the diameter of the network – leading to lower latency and lower cost. We also quantify the benefits of using high-radix routers using analysis in this chapter.

2.1 Comparing High-Radix and Low-Radix Routers

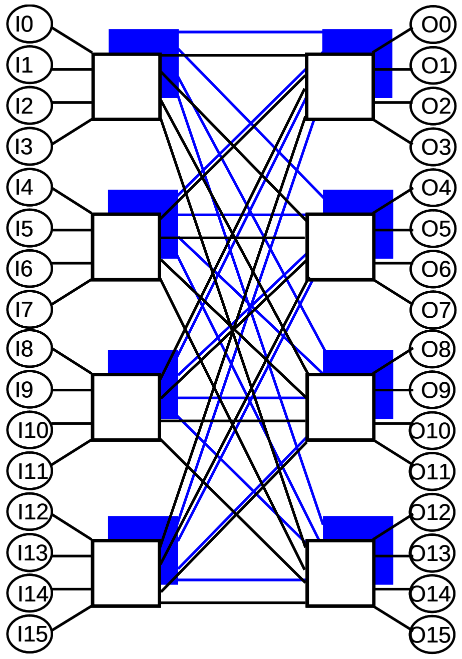
In the design of an interconnection network, the *radix* or the degree of a router significantly impacts the performance and cost of an interconnection network. With the total pin bandwidth held constant, the bandwidth can be divided among a larger number of ports to create a *high-radix* router with skinny channels or the bandwidth can be divided among a smaller number of ports to create a *low-radix* routers with fat channels.

In Figure 2.1, an example of a 16-node butterfly topology network is shown with low-radix (radix-2) routers (Figure 2.1(a)) and high-radix (radix-4) routers (Figure 2.1(b)).[1] By utilizing low-radix routers, the resulting butterfly network is a 4-stage

[1]For simplicity, radix-2 is used to illustrate low-radix and radix-4 is used to represent high-radix routers but throughout this thesis, *high* radix refers to radix-64 and higher.



(a)



(b)

Figure 2.1: 16-node butterfly network example using (a) low-radix routers and (b) high-radix routers. The high-radix network in (b) uses channel slicing to provide the same terminal bandwidth.

parameter	**low radix**	**high radix**
Latency (hop count)	4	2
Cost (# of channels)	96	32

Table 2.1: Latency and cost comparison of the two networks shown in Figure 2.1. The cost of the channels ignores the number of *terminal* channels as the cost is dominated by the *global* channels in the network.

network, while the use of high-radix routers results in only a 2-stage network and reduces the diameter of the network. The main difference between the two networks is a trade-off in the bandwidth per channel and the diameter of the network. Since the total pin bandwidth per router node is held constant, the amount of bandwidth *per* channel is different for the two networks. The low-radix networks provides 2x bandwidth per channel and thus, each terminal node receives 2x bandwidth compared to the high-radix network. In order to provide the same amount of bandwidth to the terminal nodes, the network can be duplicated to create another parallel network – a technique often referred to as *channel slicing* [22].

The two networks shown in Figure 2.1 are compared using two metrics – performance and cost. Performance is measured in terms of latency and can be approximated by the number of hop counts in the network. The cost of the network can be estimated by the number of channels in the network since the network cost is often determined by the bandwidth (or the number of channels). The comparison is summarized in Table 2.1. With the reduction in the diameter of the network by using high-radix routers, not only is the latency of the network reduced but the cost is also significantly reduced. In the following section, we provide a more analytical description of the benefits of high-radix routers.

2.2 Need for High-Radix Routers

Many of the earliest interconnection networks were designed using topologies such as butterflies or hypercubes, based on the simple observation that these topologies minimized hop count. However, different analysis in the late 80s and early 90s showed that lower-radix networks provided better performance with many studies focusing on the k-ary n-cube topology. [2]

The analysis of both Dally [19] and Agarwal [3] showed that under fixed packaging constraints, lower radix networks offered lower packet latency. Scott and Goodman [69] later extended their studies and showed that if link pipelining is used in the network, the wire latency can be decoupled from the router cycle time and as a result, it tends to favor higher dimensionality for large networks. However, Scott and Goodman also concluded that optimal dimension of a k-ary n-cube is limited since higher dimension leads to a disadvantage of reduced channel width. Today, the fundamental result of these authors still holds — technology and packaging constraints should drive topology design. What has changed in recent years are the topologies that these constraints lead us toward.

To understand how technology changes affect the optimal network radix, consider the latency (T) of a packet traveling through a network. Under low loads, this latency is the sum of header latency and serialization latency. The header latency (T_h) is the time for the beginning of a packet to traverse the network and is equal to the number of hops a packet takes times a per hop router delay (t_r). Since packets are generally wider than the network channels, the body of the packet must be squeezed across the channel, incurring an additional serialization delay (T_s). Thus, total delay can be written as

$$T = T_h + T_s = Ht_r + L/b \tag{2.1}$$

where H is the number of hops a packet travels, L is the length of a packet, and b is the bandwidth of the channels. For an N node network with radix k routers (k input channels and k output channels per router), the number of hops must be at least $2log_kN$.[3] Also, if the total bandwidth of a router is B, that bandwidth is divided among the $2k$ input and output channels and $b = B/2k$. Substituting this into the

[2]The n or the dimension of an k-ary n-cube correspond to the radix terminology that is used in this thesis. Increasing the radix reduces the diameter of the network and similarly, increasing the dimension (n) of a k-ary n-cube also reduces the diameter.

[3]Uniform traffic is assumed and $2log_kN$ hops are required for a non-blocking network.

expression for latency from Equation 2.1

$$T = 2t_r \log_k N + 2kL/B. \tag{2.2}$$

Then, From Equation 2.2, setting dT/dk equal to zero and isolating k gives the optimal radix in terms of the network parameters,

$$k \log^2 k = \frac{Bt_r \log N}{L}. \tag{2.3}$$

In this differentiation, we assume B and t_r are independent of the radix k. Since we are evaluating the optimal radix for a *given* bandwidth, we can assume B is independent of k. The t_r parameter is a function of k but has only a small impact on the total latency and has no impact on the optimal radix. Router delay t_r can be expressed as the number of pipeline stages (P) times the cycle time (t_{cy}). As radix increases, t_{cy} remains constant and P increases logarithmically. The number of pipeline stages P can be further broken down into a component that is independent of the radix (X) and a component which is dependent on the radix ($Y \log_2 k$). Thus router delay (t_r) can be rewritten as

$$t_r = t_{cy}P = t_{cy}(X + Y \log_2 k). \tag{2.4}$$

If this relationship is substituted back into Equation 2.2 and differentiated, the dependency on radix k coming from the router delay disappears and does not change the optimal radix.[4] Intuitively, although a single router delay increases with a $\log(k)$ dependence, the effect is offset in the network by the fact that the number of hop count decreases as $1/\log(k)$ and as a result, the router delay does not effect the optimal radix.

In Equation 2.2, we ignore time of flight for packets to traverse the wires that make up the network channels. The time of flight does not depend on the radix(k) and thus has minimal impact on the optimal radix. Time of flight is D/v where D is the total physical distance traveled by a packet and v is the propagation velocity.

[4]If this detailed definition of t_r is used, t_r is replaced with Xt_{cy} in Equation 2.3.

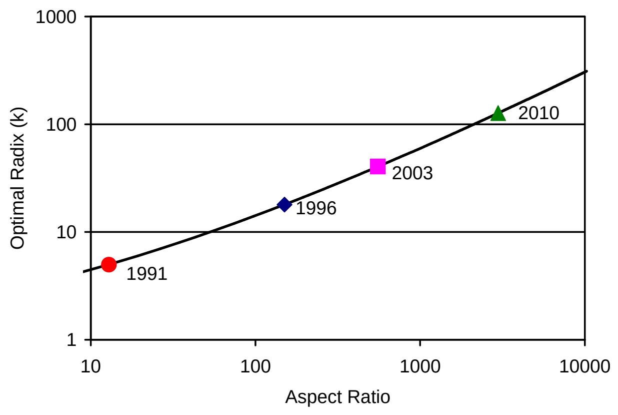


Figure 2.2: Relationship between optimal latency radix and router aspect ratio. The labeled points show the approximate aspect ratio for a given year's technology

As radix increases, the distance between two router nodes (D_{hop}) increases. However, the *total* distance traveled by a packet will be approximately equal since a lower-radix network requires more hops.

From Equation 2.3, we refer to the quantity $A = \frac{Bt_r \log N}{L}$ as the *aspect ratio* of the router. This aspect ratio completely determines the router radix that minimizes network latency. A high aspect ratio implies a "tall, skinny" router (many, narrow channels) minimizes latency, while a low ratio implies a "short, fat" router (few, wide channels).

A plot of the minimum latency radix versus aspect ratio, from Equation (2.3) is shown in Figure 2.2. The points along the line show the aspect ratios from several years. These particular numbers are representative of large supercomputers with single-word network accesses[5], but the general trend of the radix increasing significantly over time remains.

Figure 2.3(a) shows how latency varies with radix for 2003 and 2010 technologies. As radix is increased, latency first decreases as hop count, and hence T_h, is reduced. However, beyond a certain radix serialization latency begins to dominate the overall latency and latency increases. As bandwidth, and hence aspect ratio, is increased,

[5]The 1991 data is from J-Machine [60] (B=3.84Gb/s, t_r=62ns, N=1024, L=128bits), the 1996 data is from the Cray T3E [67] (64Gb/s, 40ns, 2048, 128), the 2003 data is from SGI Altix 3000 [71] (0.4Tb/s, 25ns, 1024, 128) 2010 data is estimated(20Tb/s, 5ns, 2048, 256).

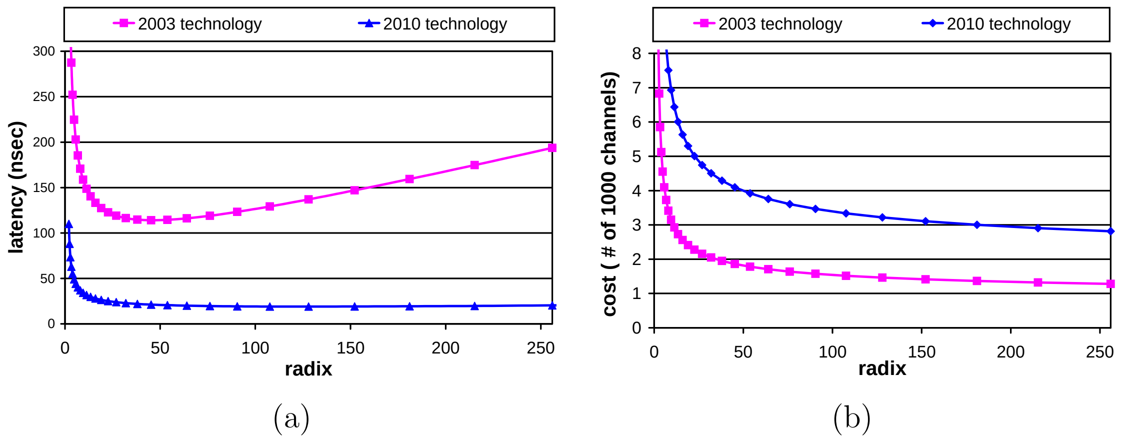


Figure 2.3: (a) Latency and (b) cost of the network as the radix is increased for two different technologies.

the radix that gives minimum latency also increases. For 2003 technology (aspect ratio = 554) the optimum radix is 40 while for 2010 technology (aspect ratio = 2978) the optimum radix is 127.

Increasing the radix of the routers in the network monotonically reduces the overall cost of a network. Network cost is largely due to router pins and connectors and hence is roughly proportional to total router bandwidth: the number of channels times their bandwidth. For a fixed network bisection bandwidth, this cost is proportional to hop count. Since increasing radix reduces hop count, higher radix networks have lower cost as shown in Figure 2.3(b).[6] Power dissipated by a network also decreases with increasing radix. Power is roughly proportional to the number of router nodes in the network. As radix increases, hop count decreases, and the number of router nodes decreases. The power of an individual router node is largely independent of the radix as long as the total router bandwidth is held constant. Router power is largely due to I/O circuits and switch bandwidth. The arbitration logic, which becomes more complex as radix increases, represents a negligible fraction of total power[79].

[6]2010 technology is shown to have higher cost than 2003 technology because the number of nodes is much greater.

2.3 Summary

As technology evolves, its impact on interconnection networks needs to be re-evaluated. With the increasing pin bandwidth available today, we have shown how the use of high-radix routers exploits increased pin bandwidth by reducing the diameter of the network – resulting in lower latency and lower cost. In the following chapter, we will first look at an efficient topology that can take advantage of high-radix routers.

Chapter 3

Flattened Butterfly Topology

In this chapter, we first discuss existing topologies which can exploit high-radix routers and their shortcomings. Then, we describe a cost-efficient topology that we refer to as the *flattened butterfly* topology and how it can be derived from the conventional butterfly. The advantages of the flattened butterfly compared to alternative topologies are described. In the final section of this chapter, we describe a detailed cost model for interconnection networks and provide a cost comparison of the flattened butterfly to alternative topologies.

3.1 Existing High-Radix Topologies

The minimum diameter (D), the largest hop count between any two terminals, of the network is

$$D \geq log_k N \tag{3.1}$$

where N is the size of the network (or the number of terminals in the network) and k is the radix of the routers. The butterfly network (k-ary n-fly) can achieve this minimum diameter and take advantage of high-radix routers to reduce latency and network cost [22] as shown in Figure 3.1. [1] However, there is no path diversity in a

[1]For simplicity, the diagrams are drawn with radix-2 (low-radix) routers but this thesis deals with high-radix routers with radix 64 or higher.

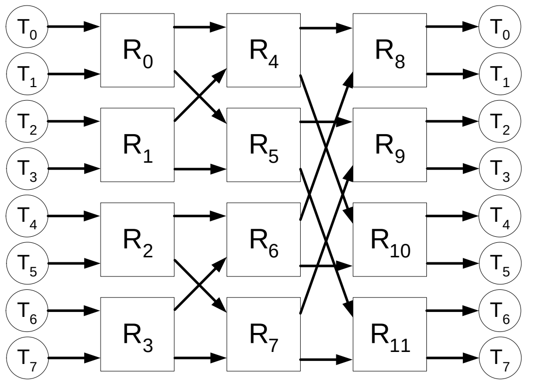


Figure 3.1: Block diagram of a conventional butterfly topology with 3-stages.

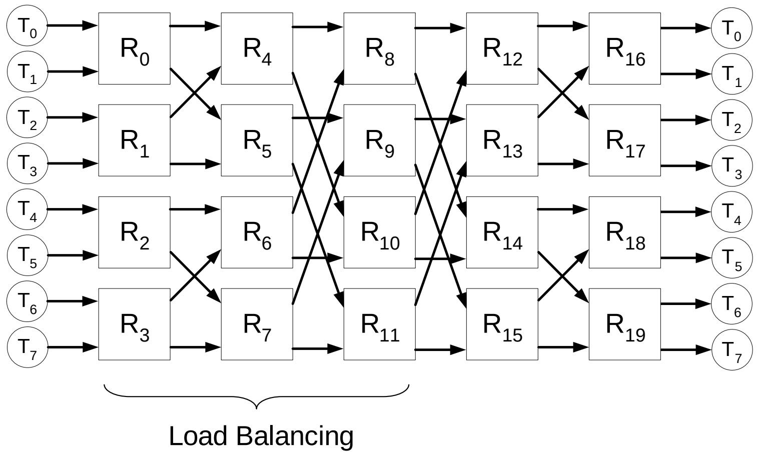


Figure 3.2: Block diagram of a Clos topology.

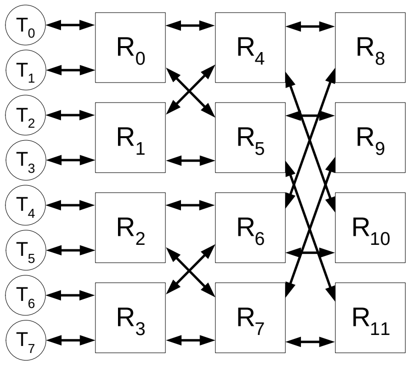


Figure 3.3: Block diagram of a Folded-Clos (fat-tree) topology.

butterfly network which results in poor throughput for adversarial traffic patterns. For example in Figure 3.1, for a particular traffic pattern where T0 sends traffic to T1, and T4 sends traffic to T0, both of these traffic patterns will have to share the bandwidth for channel (R4,R8) – resulting in a throughput degradation. In the worst-case traffic pattern, the throughput of the butterfly network can be reduced by a factor of $\sqrt{N}$. Also, a butterfly network cannot exploit traffic locality since every packet needs to traverse D hop counts.

To add path diversity to a butterfly network and provide better performance, two butterfly networks can be connected back-to-back to create a Clos network [17] as shown in Figure 3.2. The first half of the Clos network is used for load balancing and thus, provides many paths [2] between each pair of terminals. This path diversity enables the Clos to route arbitrary traffic patterns with no loss of throughput. The input and output stages of a Clos network can be combined or *folded* on top of one another creating a folded-Clos or fat-tree [52] network which can exploit traffic locality (Figure 3.3).

A Clos or folded-Clos network, however, has a cost that is nearly double that of a butterfly network with equal capacity and also has greater latency than a butterfly network. The increased cost and latency both stem from the need to route packets first to an arbitrary middle stage switch and then to their ultimate destination. This doubles the number of long cables in the network, which approximately doubles the cost, and doubles the number of inter-router channels traversed, which drives up latency.

The flattened butterfly topology exploits high-radix routers to achieve lower cost than a folded-Clos on load-balanced traffic, and provides better performance and path diversity than a conventional butterfly. In this chapter, we describe the flattened butterfly topology in detail by comparing it to a conventional butterfly and show how it is also similar to a folded-Clos.

[2] One for each middle-stage switch in the Clos.

3.2 Topology Construction: Butterfly to Flattened Butterfly

We derive the flattened butterfly by starting with a conventional butterfly (k-ary n-fly) and combining or *flattening* the routers in each *row* of the network into a single router. Examples of flattened butterfly construction are shown in Figure 3.4. 4-ary 2-fly and 2-ary 4-fly networks are shown in Figure 3.4(a,c) with the corresponding flattened butterflies shown in Figure 3.4(b,d). The routers R0 and R1 from the first row of Figure 3.4(a) are combined into a single router R0′ in the flattened butterfly of Figure 3.4(b). Similarly, routers R0, R1, R2, and R3 of Figure 3.4(c) are combined into R0′ of Figure 3.4(d). As a row of routers is combined, channels entirely local to the row, e.g., channel (R0,R1) in Figure 3.4(a), are eliminated. All other channels of the original butterfly remain in the flattened butterfly. For example, channel (R0,R3) in Figure 3.4(a) becomes channel (R0′,R1′) in Figure 3.4(b). Because channels in a flattened butterfly are symmetrical, each line in Figures 3.4(b,d) represents a bidirectional channel (i.e. two unidirectional channels), while each line in Figures 3.4(a,c) represents a unidirectional channel.

A k-ary n-flat, the flattened butterfly derived from a k-ary n-fly, is composed of $\frac{N}{k}$ radix $k' = n(k-1)+1$ routers where N is the size of the network. The routers are connected by channels in $n' = n-1$ dimensions, corresponding to the $n-1$ columns of inter-rank wiring in the butterfly. The lowest dimension corresponds to the right most column of inter-rank wiring and the corresponding dimension increases as the column moves from right to left. In each dimension d, from 1 to n', router i is connected to each router j given by

$$j = i + \left[m - \left(\left\lfloor \frac{i}{k^{d-1}} \right\rfloor mod\ k\right)\right] k^{d-1} \tag{3.2}$$

for m from 0 to $k-1$, where the connection from i to itself is omitted. For example, in Figure 3.4(d), R4′ is connected to R5′ in dimension 1, R6′ in dimension 2, and R0′ in dimension 3.

The number of nodes (N) in a flattened butterfly is plotted as a function of

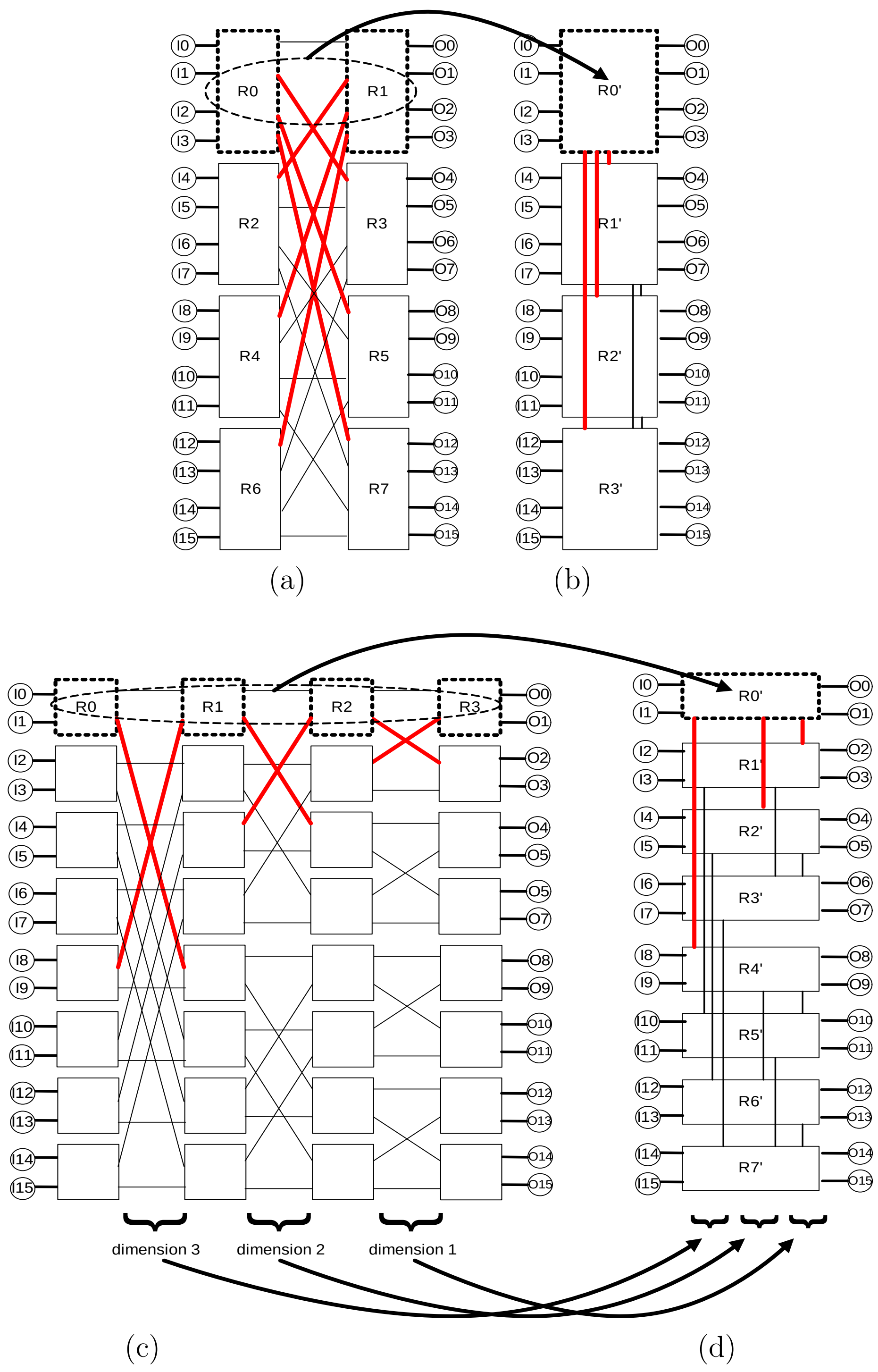

I0 I1 I2 I3 I4 I5 I6 I7 I8 I9 I10 I11 I12 I13 I14 I15
O0 O1 O2 O3 O4 O5 O6 O7 O8 O9 O10 O11 O12 O13 O14 O15
R0
R1
R2
R3
R4
R5
R6
R7
R0'
R1'
R2'
R3'
R4'
R5'
R6'
R7'
dimension 3
dimension 2
dimension 1
(a)
(b)
(c)
(d)

Figure 3.4: Block diagram of (a) 4-ary 2-fly butterfly and (b) 4-ary 2-flat – the corresponding flattened butterfly with a single dimension, (c) 2-ary 4-fly butterfly and (d) 2-ary 4-flat – the corresponding flattened butterfly with three dimensions.

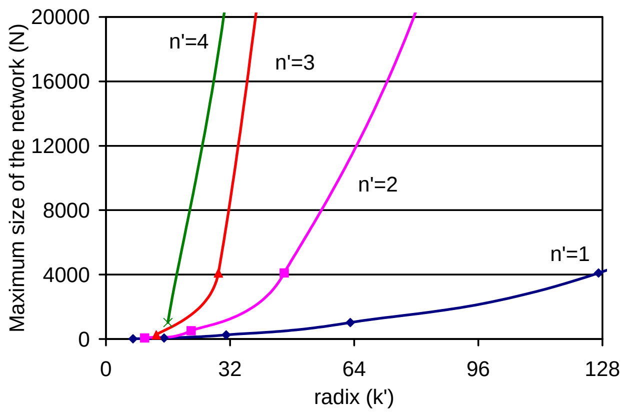


Figure 3.5: Network size (N) scalability as the radix (k') and dimension (n') is varied in a flattened butterfly.

number of dimensions n' and switch radix k' in Figure 3.5. The figure shows that this topology is suited only for high-radix routers. Networks of very limited size can be built using low-radix routers ($k' < 16$) and even with $k' = 32$, many dimensions are needed to scale to large network sizes. However with $k' = 61$, a network with just three dimensions scales to 64K nodes.

3.3 Routing and Path Diversity

Routing in a flattened butterfly requires a hop from a node to its local router, zero or more inter-router hops, and a final hop from a router to the destination node. If we label each node with a n-digit radix-k node address, an inter-router hop in dimension d changes the d^{th} digit of the current node address to an arbitrary value, and the final hop sets the 0^{th} (rightmost) digit of the current node address to an arbitrary value. Thus, to route minimally from node $a = a_{n-1}, \ldots, a_0$ to node $b = b_{n-1}, \ldots, b_0$ where a and b are n-digit radix-k node addresses involves taking one inter-router hop for each digit, other than the rightmost, in which a and b differ. For example, in Figure 3.4(d) routing from node 0 (0000_2) to node 10 (1010_2) requires taking inter-router hops in dimensions 1 and 3. These inter-router hops can be taken in either order giving two minimal routes between these two nodes. In general, if two nodes a and b have addresses that differ in j digits (other than the rightmost digit), then there are $j!$ minimal routes between a and b. This path diversity derives from the fact that a packet routing in a flattened butterfly is able to traverse the dimensions in any order, while a packet traversing a conventional butterfly must visit the dimensions in a fixed order – leading to no path diversity.

Routing non-minimally in a flattened butterfly provides additional path diversity and can achieve load-balanced routing for arbitrary traffic patterns. Consider, for example, Figure 3.4(b) and suppose that all of the traffic from nodes 0-3 (attached to router R0′) was destined for nodes 4-7 (attached to R1′). With minimal routing, all of this traffic would overload channel (R0′,R1′). By misrouting a fraction of this traffic to R2′ and R3′, which then forward the traffic on to R1′, load is balanced. With non-minimal routing, [3] a flattened butterfly is able to match the load-balancing (and non-blocking) properties of a Clos network – in effect acting as a *flattened* folded-Clos. To illustrate how the flattened butterfly can be also referred to as a flattened folded-Clos, a 3-stage folded-Clos is shown in Figure 3.6(a). The routers in each row of the folded-Clos, as highlighted with a dotted box, can be combined to generate a *flattened* folded-Clos as shown in Figure 3.6(b). With only one middle stage, the resulting flattened folded-Clos is identical to a one dimensional flattened butterfly but increasing the number of stages of folded-Clos corresponds to higher dimension flattened butterfly.

3.4 Cost Model

In this section, we compare the cost of the flattened butterfly topology to three alternative high-radix topologies – conventional butterfly, folded-Clos, and hypercube. We first present a cost model which takes into account the main components of an interconnection network, the routers and the cables, and incorporates the packaging hierarchy of the network into the cost model. We also describe the packaging of the

[3] We discuss non-minimal routing strategies for flattened butterfly networks in more detail in Section 4.2.

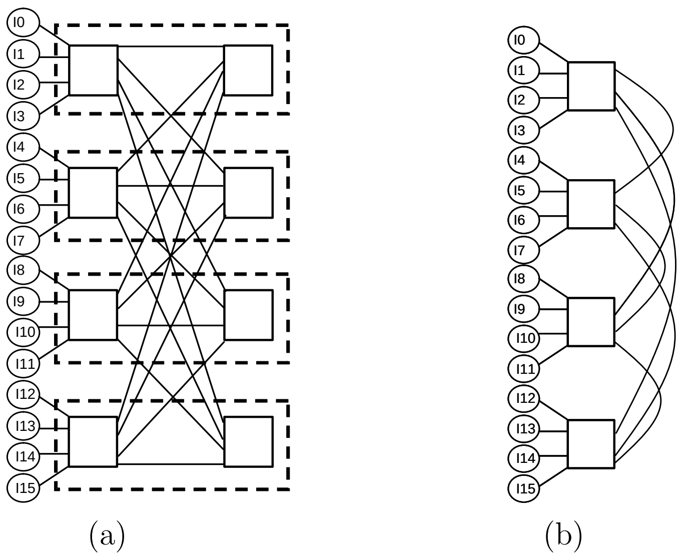


Figure 3.6: Block diagram of (a) a folded-Clos and the routers in each row as outlined in the dotted box are combined to create a corresponding (b) *flattened* Clos.

topologies and use them to estimate the length of the cables. Using our cost model, we compare the cost of the different topologies and describe the cost advantages of the flattened butterfly. In the cost comparison, we hold the performance on load-balanced traffic (i.e. bisection bandwidth) constant to provide a fair comparison.

3.4.1 Cost Model

The system components, processing nodes and routers, are packaged within a *packaging hierarchy*. At the lowest level of the hierarchy are the compute modules (containing the processing nodes) and routing modules (containing the routers). At the next level of the hierarchy, the modules are connected via a backplane or midplane printed circuit board. The modules and backplane are contained within a *cabinet*. A *system* consists of one or more cabinets with the necessary *cables* connecting the router ports according to the network topology. The network cables may aggregate multiple network links into a single cable to reduce cost and cable bulk.

Component	**Cost**
router	\$390
router chip	\$90
development	\$300
links (cost per signal 6Gb/s)	
backplane	\$1.95
electrical	\$3.72 + \$0.81 ℓ
optical	\$220.00

Table 3.1: Cost breakdown of an interconnection network.

Network cost is determined by the cost of the routers and the backplane and cable links. The cost of a link depends on its length and location within the packaging hierarchy (Table 3.1). In 2007, a link within the backplane[4] is about \$1.95 per differential signal[5], whereas a cable connecting nearby routers (routers within 2m) is about \$5.34 per signal, and an optical cable is about \$220 per signal[6] to connect routers in a distant cabinet. Inexpensive backplanes are used to connect modules in the same cabinet over distances that are typically less than 1m. Moderate cost electrical cables connect modules in different cabinets up to lengths of 5-10m.[7] Transmitting signals over longer distance require either an expensive optical cable or a series of electrical cables connected by *repeaters* that retime and amplify the signal. Because optical technology still remains relatively expensive compared to electrical signaling over copper, our cost analysis uses electrical signaling with repeaters as necessary for driving signals over long distances.

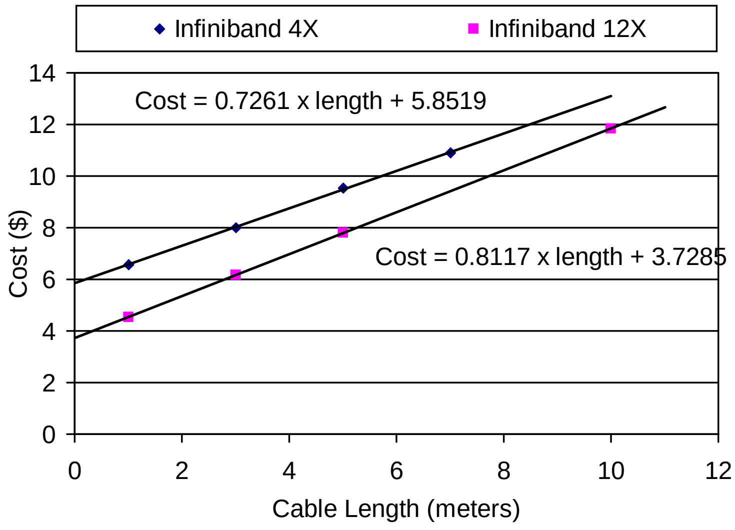
Router cost is divided into development (non-recurring) cost and silicon (recurring) cost. The development cost is amortized over the number of router chips built.

[4]We assume \$3000 backplane cost for 1536 signals, or \$1.95 per backplane signal which includes the GbX connector cost at \$0.12 per mated signal to support signal rates up to 10 Gb/s [6].

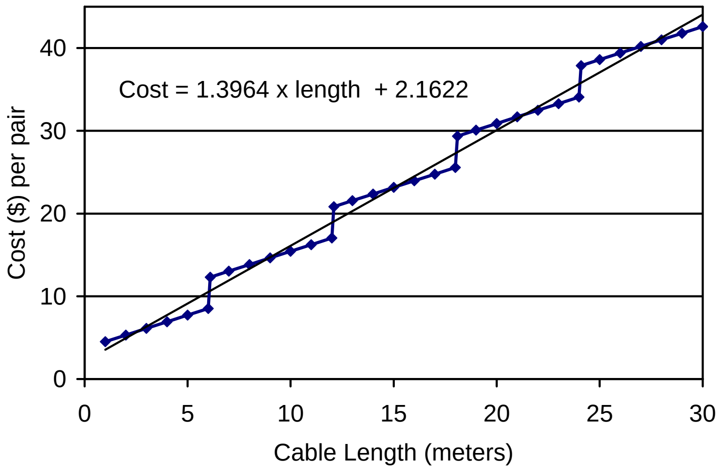
[5]To provide signal integrity at high speed, signals are almost always transmitted differentially - using a pair of wires per signal.

[6]Pricing from www.boxfire.com – \$480 for 20 meter 24 strand terminated fiber optic cable results in \$20 per signal for the cost of the cable. Although optical transceiver modules have a price goal of around \$200, that has not yet been achieved with current modules costing \$1200 and up.

[7]In our analysis, we will assume that a 6m cable is the longest distance that can be driven at full signalling rate of 6.25 Gb/s, based on SerDes technology similar to that used in the Cray BlackWidow [68].



(a)



(b)

Figure 3.7: Cable Cost Data (2007). (a) Cost of Infiniband 4x and 12x cables as a function of cable length and (b) cable cost model with the use of repeaters for cable >6m. The model is based on the Infiniband 12x cost model and the data point is fitted with a line to calculate the average cable cost.

We assume a non-recurring development cost of ≈\$6M for the router which is amortized over 20k parts, about \$300 per router. The recurring cost is the cost for each silicon part which we assume to be ≈\$90 per router using the MPR cost model [55] for a TSMC 0.13μm 17mm×17mm chip which includes the cost of packaging and test.

The cost of electrical cables can be divided into the cost of the wires (copper) and the overhead cost which includes the connectors, shielding, and cable assembly. Figure 3.7(a) plots the cost per differential signal for Infiniband 4x and 12x cables as a function of distance. The cost of the cables was obtained from Gore [30]. The slope of the line represents the cable cost per unit length (\$/meter) and the y-intercept is the overhead cost associated with shielding, assembly and test. With the Infiniband 12x cables, the overhead cost per signal is reduced by 36% because of cable aggregation – Infiniband 12x contains 24 wire pairs while Infiniband 4x contains only 8 pairs, thus 12x cable is able to amortize the shielding and assembly cost. [8]

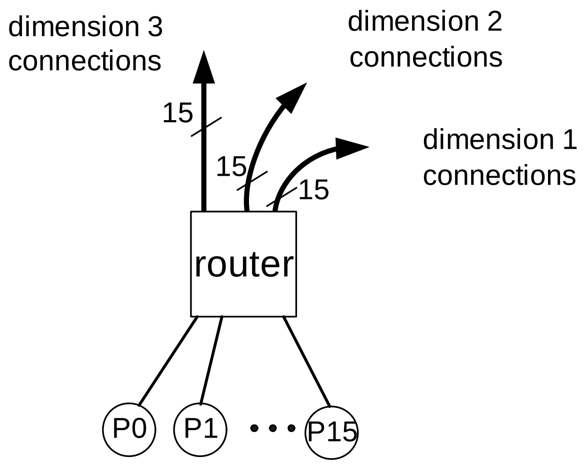
When the cable length exceeds the critical length of 6m, repeaters need to be inserted and the cable cost with repeaters is shown in Figure 3.7(b), based on the Infiniband 12x cable cost. When length is 6m, there is a step in the cost which reflects the repeater cost. Since the cost of the repeaters is dominated by the extra connector cost, the increase in the cost function is approximately the additional connector cost. The linear cost model shown in Figure 3.7(b) is used for all cables.

3.4.2 Packaging and Cable length

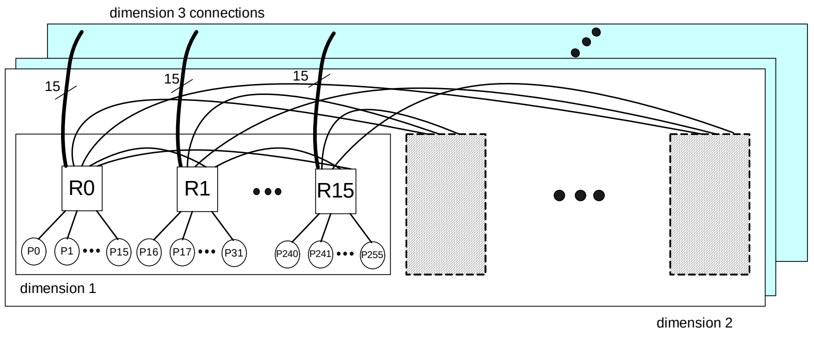
Figure 3.8 shows a possible packaging of a 16-ary 4-flat ($k' = 61$, $n' = 3$) flattened butterfly network. Each router has channels to 16 terminal nodes and to 45 other routers, 15 in each of three dimensions (Figure 3.8(a)). Figure 3.8(b) shows how the routers connected in dimension 1 are packaged in a subsystem containing 256 nodes.[9] The like elements from 16 of these dimension 1 subsystems are connected in

[8]It is interesting to note that the 12x has a slightly higher wire cost per unit length. The Infiniband 4x is a *commodity* cable whereas the 12x cables needed to be custom ordered – thus, resulting in the higher price.

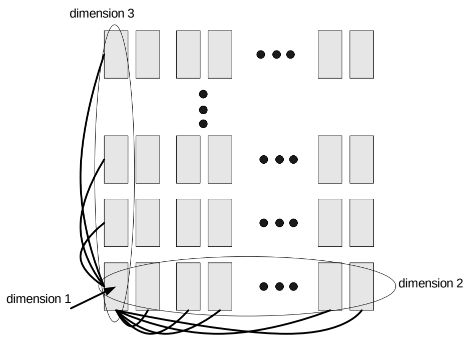
[9]We assume packaging 128 nodes per cabinet as in Cray BlackWidow. Hence a dimension 1 subsystem requires two cabinets connected by short cables.



(a)



(b)



(c)

Figure 3.8: Block diagrams of a 16-ary 4-flat flattened butterfly. (a) Block diagram of each router where 16 ports are used for the terminal nodes and 15 ports are used for connections in each dimension. (b) Topology block diagram that can scale to more than 64K nodes. (c) A packaging block diagram of the topology, with the connections for only the lower left cabinet shown.

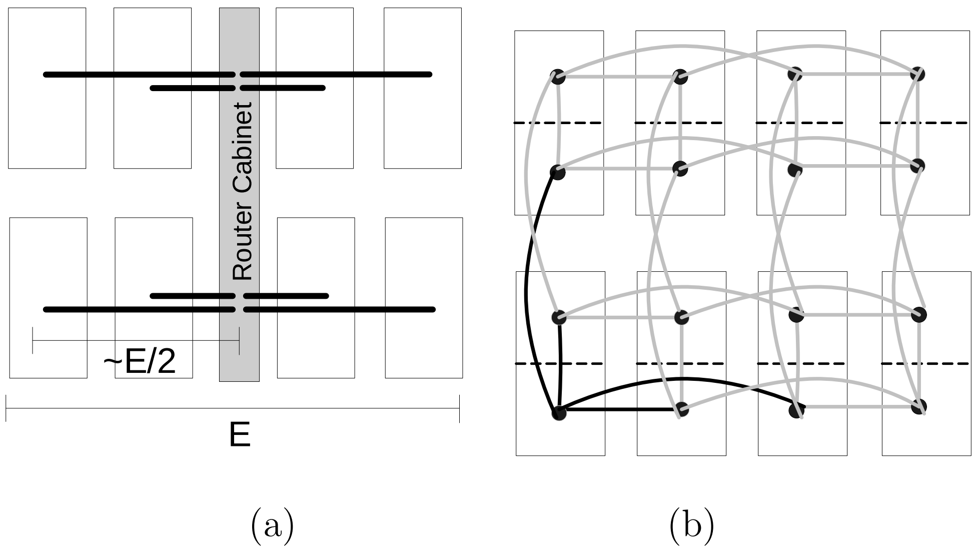


Figure 3.9: Sample cabinet packaging layout of 1024 nodes in (a) folded-Clos and (b) hypercube topologies. Each box represents a cabinet of 128 nodes and the hypercube is partitioned into two chassis. In the folded-Clos, the router cabinet is assumed to be placed in the middle and for illustration purpose only, the router cabinet is drawn taller.

dimensions 2 forming a subsystem with 4,096 nodes. Up to 16 of these subsystems can be combined in dimension 3 leading to a network with up to 65,536 nodes. A possible packaging of this configuration is shown in Figure 3.8(c) where each box represents a pair of cabinets that contains a dimension 1 subsystem. Dimension 2 is mapped across the columns, and dimension 3 is mapped across the rows. The connections are only shown for the lower left cabinet.

From Figure 3.8(c), the maximum cable length (L_{max}) of the flattened butterfly topology will be approximately equal to the length of one edge (E) of the 2-D cabinet layout. The average *global* cable length (L_{avg}) for connections in dimensions 2 and 3 is approximately $L_{max}/3 = E/3$. Dimension 1 connections between nodes are made over backplanes or very short (1-2m) cables between nodes in different cabinets. It can be seen that the L_{max} and the L_{avg} of the butterfly will be the same as that of the flattened butterfly since the flattened butterfly's channels were derived from the channels in the conventional butterfly.

Packaging of the folded-Clos and the hypercube are shown in Figure 3.9. For the folded-Clos, L_{max} is approximately $E/2$ since the cables only need to be routed to a central routing cabinet (Figure 3.9(a)). L_{avg} for the Clos is approximately $L_{max}/2 = E/4$. The hypercube has similar L_{max} cable lengths as the folded-Clos (Figure 3.9(b)). The diagram illustrates only the *global* cables – inter-cabinet connections for the higher dimensions in the hypercube. Each node of the hypercube connects to a single node in each dimension of the network, thus the longest cable will be $E/2$, the next longest cable will be $E/4$ and so forth. The cable lengths are a geometric distribution, which can be summed to arrive at the average cable length, L_{avg} of approximately $(E-1)/log_2(E)$. Because of the logarithmic term, as the network size increases, the average cable length is shorter than the other topologies.

The length of an edge (E) in a cabinet packaging layout can be estimated as

$$E = \sqrt{\frac{N}{D}}. \tag{3.3}$$

where N is the number of nodes and D is the density of nodes (nodes/m^2).

3.4.3 Cost Comparison

The assumptions and the parameters used in the topology cost comparison are shown in Table 3.2. In determining the actual cable length for both L_{max} and L_{avg}, we add 2m of cable overhead – 1m of vertical cable run at each end of the cable. We multiply a factor of 2 to the depth of the cabinet footprint to allow for spacing between rows of cabinets. Based on these parameters and the cost model described earlier, we compare the cost of conventional butterfly, folded-Clos, hypercube, and flattened butterfly topologies while holding the random bisection bandwidth constant.

In Figure 3.10, the ratio of the link cost to the total interconnection network cost is shown. The total cost of the interconnection network is dominated by the cost of the links. Because of the number of routers in the hypercube, the routers dominate the cost for small configurations.[10] However, for larger hypercube networks

[10]The router cost for the hypercube is appropriately adjusted, based on the number of pins required. Using concentration in the hypercube could reduce the cost of the network but will significantly degrade performance on adversarial traffic pattern and thus, is not considered.

parameter	**value**
radix	64
# of pairs per port	3
nodes per cabinet	128
cabinet footprint	0.57m x 1.44m
D (density)	75 nodes/m^2
cable overhead	2m

Table 3.2: Technology and packaging assumptions used in the topology comparison. The values are representative of those used in the Cray BlackWidow parallel computer.

($N > 4K$), link cost accounts for approximately 60% of network cost. For the other three topologies, link cost accounts for approximately 80% of network cost. Thus, it is critical to reduce the number of links to reduce the cost of the interconnection network.

The average cable length (L_{avg}) of the different topologies is plotted in Figure 3.11) as the network size is varied, based on the cable length model presented in Section 3.4.2. For small network size (<4K), there is very little difference in L_{avg} among the different topologies. For larger network size, L_{avg} for the flattened butterfly is 22% longer than that of a folded-Clos and 54% longer than that of a hypercube. However, with the reduction in the number of global cables, the flattened butterfly is still able to achieve a cost reduction compared to the other topologies.

We plot the cost per node of the four topologies in Figure 3.12 as the network size is increased. In general, the butterfly network provides the lowest cost while the hypercube and the folded-Clos have the highest cost. The flattened butterfly, by reducing the number of links, gives a 35-53% reduction in cost compared to the folded-Clos.

A step increase occurs in the cost of the folded-Clos when the number of stages increases. For example, increasing the network size from 1K to 2K nodes with radix-64 routers requires adding an additional stage to the network to go from a 2-stage folded-Clos to a 3-stage folded-Clos. The flattened butterfly has a similar *step* in the cost function when the number of dimensions increases. For example, a dimension

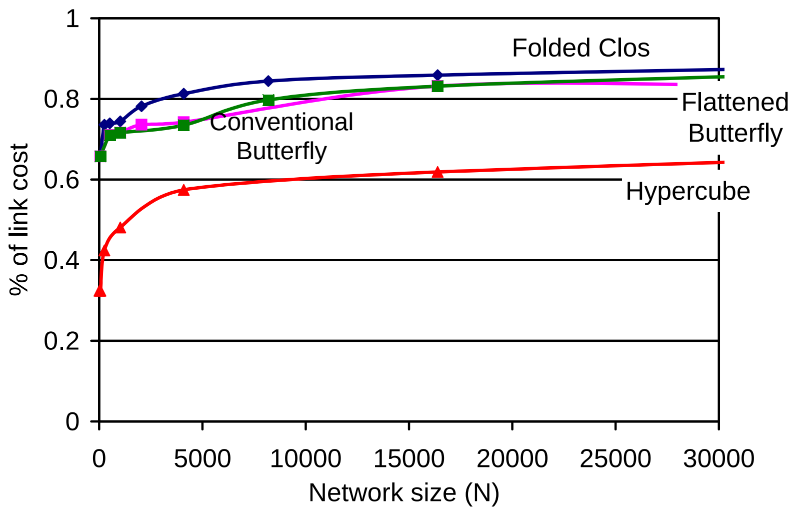


Figure 3.10: The ratio of the link cost to total network cost. The cable overhead is not included in this plot.

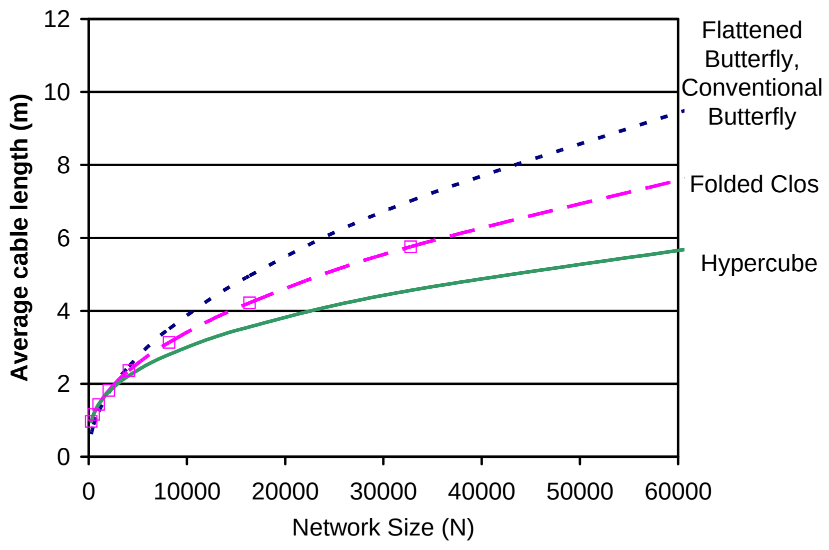


Figure 3.11: Average cable length of the different topologies as the network size is increased. The cable overhead is not included in this plot.

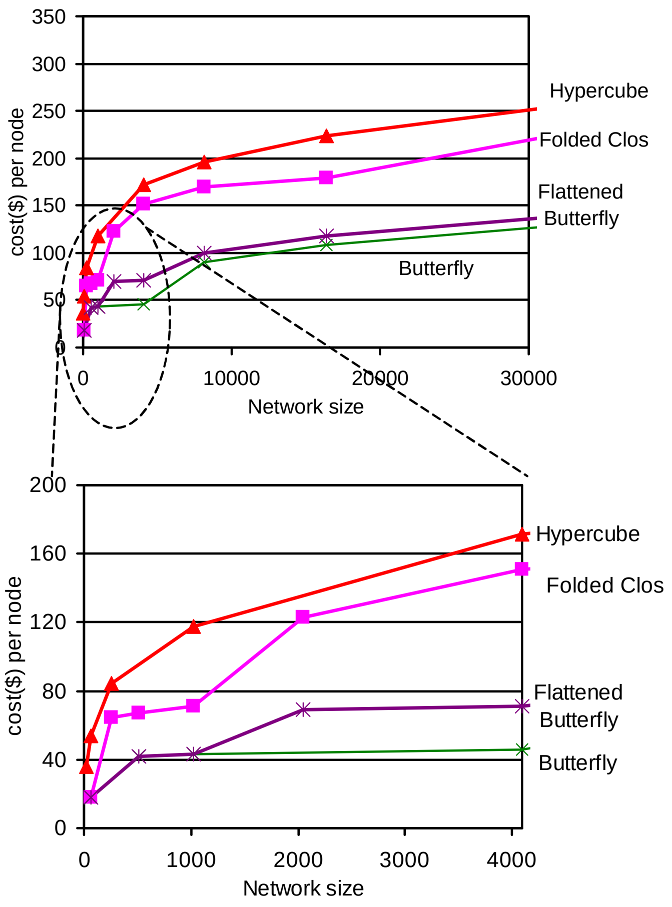


Figure 3.12: Cost comparison of the different topologies. The bottom plot is the same plot as the plot on the top with the x-axis zoomed in to display the smaller networks more clearly.

must be added to the network to scale from 1K to 2K nodes. However, the cost increase is much smaller (approximately \$40 per node increase compared to \$60 per node increase for the folded-Clos) since only a single link is added by increasing the number of dimension – whereas in the folded-Clos, two links are added.

For small networks (<1K), the cost benefit of flattened butterfly compared to the folded-Clos is approximately 35% to 38%. Although the flattened butterfly halves the number of *global* cables, it does not reduce the number of local links from the processors to the routers. For small networks, these links account for approximately

40% of the total cost per node – thus, total reduction in cost is less than the expected 50%.

For larger networks (>1K), the cost benefit is greater than 40% and at $N = 4$K, the cost is reduced by about 53%. The cost reduction of more than 50% is the result of the *packaging locality* that can be exploited with the flattened butterfly. With the flattened butterfly, dimension 1 connections are contained within a pair of adjacent cabinets and made with short links. In the folded-Clos, however, the corresponding links are routed to a central cabinet requiring *global* links. In addition, the number of links are actually reduced by more than 1/2 in the flattened butterfly – for example, with $N = 1$K network, the folded-Clos requires 2048 links while the flattened butterfly requires 31 x 32 = 992 links, not 1024 links. For even larger network sizes (16K-32K), the cost benefit reduces to 40 - 45% since the flattened butterfly requires higher average cable length as shown in Figure 3.11.

Although the flattened butterfly was constructed from the conventional butterfly, the conventional butterfly is a lower cost network for 1K $< N <$ 4K. With radix-64 routers, the conventional butterfly can scale to 4K nodes with only 2 stages (e.g. only one inter-router link per node). Because the flattened butterfly shares the radix of its router across stages (dimensions), it has a smaller effective radix (e.g., $k = 16$ for $k' = 61$) resulting in more stages for the same number of nodes. However, when $N >$ 4K, the butterfly requires 3 stages with all of the inter-router links being *global* – thus, the cost of the flattened butterfly becomes very comparable to the conventional butterfly.

3.5 Discussion

This section discusses trade-offs in the design parameters of the flattened butterfly, the impact of wire delay, and provides a power consumption comparison of the alternative high-radix topologies which include hypercube, folded-Clos, and the conventional butterfly networks.

k	n	k'	n'
64	2	127	1
16	3	46	2
8	4	29	3
4	6	19	5
2	12	12	11

Table 3.3: Different k and n parameters for a $N = 4$K network and the corresponding k' and n' flattened butterfly parameter.

3.5.1 Design Considerations

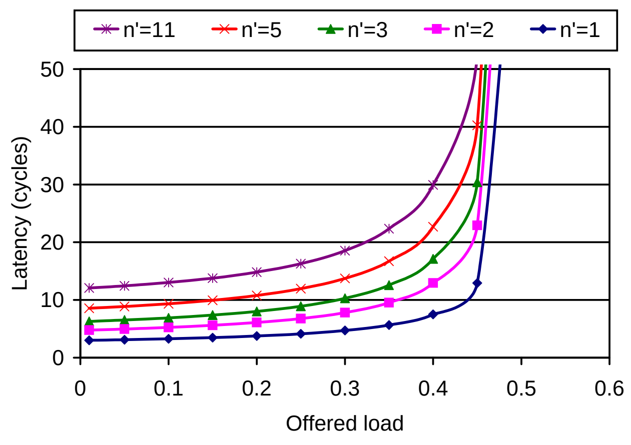
Fixed Network Size (N)

For a network with $N = 4$K nodes, Table 3.3 shows several values of k' and n' along with the corresponding values of k and n. The performance of the different configurations on uniform random (UR) traffic using the Valiant's (VAL) routing algorithm [77] is shown in Figure 3.13(a). [11] As k' decreases, the diameter of the network and hence latency increases. Because the bisection bandwidth is constant across configurations, throughput remains constant at 50% of capacity.

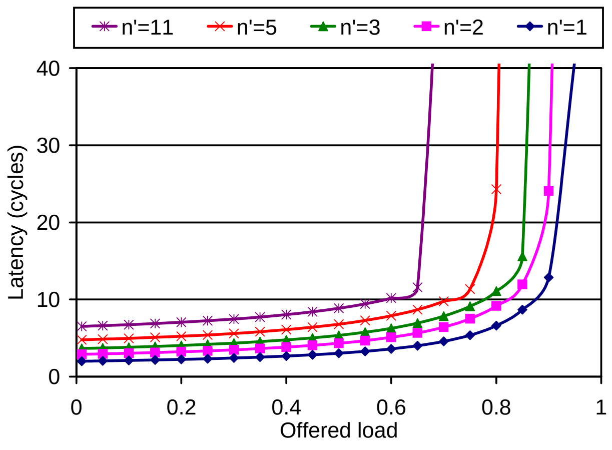
An additional affect of increased dimensionality is the increase in the number of virtual channels needed to avoid deadlock. With VAL, all of the networks require only 2 VCs to avoid deadlock. However, with adaptive routing, the number of VCs needed is proportional to n'. Figure 3.13(b) compares the performance of the different configurations using minimal adaptive (MIN AD) routing with the total storage per physical channel (PC) held constant at 64 flit buffers. As with VAL, the networks with higher n' have higher latency. However, since the total storage per PC is divided among n' VCs, increasing n' decreases the storage per VC and hence decreases throughput. The figure shows that throughput degrades about 20% as n' is increased from 1 to 5.

Using the same cost model described in Section 3.4, the cost per node is compared for the different configurations in Figure 3.14. As n' increases, the average cable length

[11]Details of alternative routing algorithms on the flattened butterfly topology is presented in Section 4.2.1 and the simulation setup is described in Section 4.2.2.



(a)



(b)

Figure 3.13: Performance comparison for different $N = 4\text{K}$ Flattened Butterflies using (a) VAL and (b) MIN AD routing algorithms.

and hence average cost per cable, decreases as shown in the line plot of Figure 3.14. However, this decrease in cost per cable is more than offset by the increase in the number of links and routers as n' increases. The cost per node increase by 45% from $n' = 1$ to $n' = 2$ and increases by 300% from $n' = 1$ to $n' = 5$. Thus, for a given N, the highest radix (and lowest dimensionality) that can be realized results in the highest performance and lowest cost.

Fixed Radix (k)

To build a flattened butterfly topology with radix-k routers, the smallest dimension (n') of the flattened butterfly should be selected that meets the scaling requirement of the network – e.g.

$$\left\lfloor \frac{k}{n'+1} \right\rfloor^{(n'+1)} \geq N. \tag{3.4}$$

Based on the value of n' selected, the resulting effective radix (k') of the topology is

$$k' = \left(\left\lfloor \frac{k}{n'+1} \right\rfloor - 1 \right)(n'+1) + 1. \tag{3.5}$$

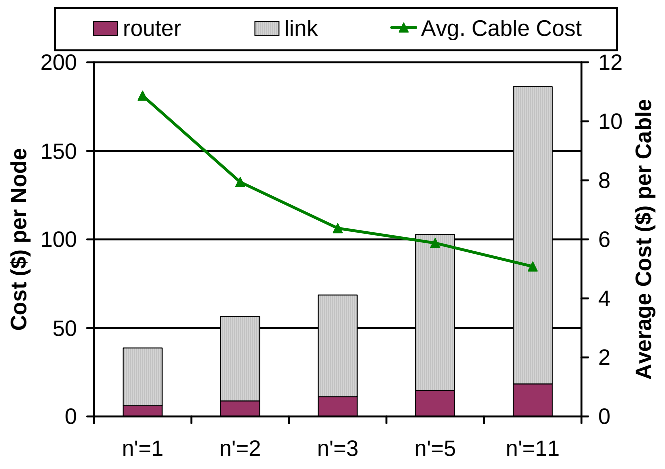


Figure 3.14: Cost Comparison of $N = 4K$ Flattened Butterflies as n' is increased.

However, depending on value of n' selected, k' may be smaller than k – thus providing extra ports in the router. For example, with radix-64 routers, a flattened butterfly with $n' = 1$ only requires $k' = 63$ to scale to 1K nodes and with $n' = 3$ only requires $k' = 61$ to scale to 64K nodes. The extra ports can be used to increase the size of the network or can be used as redundant ports. An example of expanding a 4-ary 2-flat using radix-8 routers are shown in Figure 3.15. Since 4-ary 2-flat requires radix-7 routers, an extra port can be used to increase the scalability from $N = 16$ to $N = 20$ (Figure 3.15(b)). The extra port can also be used to double the bandwidth between local router nodes (Figure 3.15(a)) which can increase the bandwidth to neighboring router nodes. However, taking advantage of the extra ports does not fundamentally change any characteristics of the topology. The use of redundant links will add some additional cost to the network but the cost advantage of the flattened butterfly and the need for global adaptive routing on the topology are still applicable.

3.5.2 Wire Delay

As shown in Figure 3.11, the flattened butterfly increases the average length of the global cables and can increase the wire delay. However, longer wire delay does not necessarily lead to longer overall latency since the wire delay (time of flight) is based on the physical distance between a source and its destination. As long as the packaging

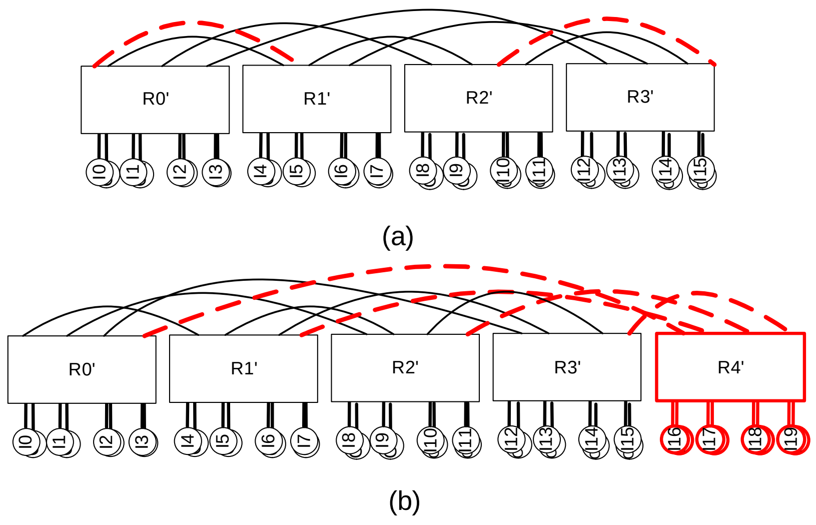


Figure 3.15: Examples of alternative organization of a 4-ary 2-flat flattened butterfly by (a) using redundant channels and (b) increasing the scalability. The dotted lines represent the additional links added by taking advantage of the extra ports.

of the topology have *minimal* physical distance, the time of flight is approximately the same regardless of the hop count. Direct networks such as torus or hypercube have minimal physical distance between two nodes but indirect networks such as folded-Clos and the conventional butterfly have non-minimal physical distance that can add additional wire latency. However, since there are no intermediate stages in the flattened butterfly, the packaging is similar to a direct network with minimal[12] distance physical distance and does not increase the overall wire delay. For this reason, the wire delay of a flattened butterfly may be smaller than that of an equivalent folded-Clos. For local traffic such as a traffic pattern where each node associated with router R_i sends traffic to a node associated with router R_{i+1}, the folded-Clos needs to route through middle stages – incurring 2x global wire delay where as the flattened butterfly can take advantage of the packaging locality and the minimum physical distance to provide lower delay.

[12]Minimal distance here means minimal *Manhattan* distance and not the distance of a straight line between the two endpoints.

3.5.3 Power comparison

The power consumption of an interconnect network is an increasing concern. The total power consumption of an interconnection network is the sum of two components: P_{switch} and P_{link}. P_{switch} is the power consumed internal to the router, including the switch, the arbitration logic, and the routing logic, and is proportional to the total *bandwidth* of the router. The complexity of arbitration and routing is increased with the larger number of ports in a high-radix router but Wang et al. has shown that these components have negligible impact on the overall power [79]. P_{link} is the power required to drive the links between the routers and is often the dominating component of the total power consumed by an interconnection network. In the Avici TSR Router [20], 60% of the power budget in the line card is dedicated to the link circuitry and approximately 45% of the power in YARC router [68] is consumed by the serializer/deserializer(SerDes). Depending on the medium (e.g. cables, backplane, etc), P_{link} can vary significantly. Using the same SerDes, the power consumed to drive a local link (P_{link_gl}) is 20% less than the power consumed to drive a global cable (P_{link_gg}) with the reduction in power coming from the equalizer and the transmitter/receiver [13]. For a router in an indirect topology, depending on where in the topology the router is used, a SerDes might need to drive a local link or a global link.

However, for direct topologies and for the flattened butterfly, a dedicated SerDes can be used to exploit the packaging locality – e.g. have separate SerDes on the router chip where some of the SerDes are used to drive local links while others are used to drive global links and reduce the power consumption. For example, a SerDes that can drive <1m of backplane only consumes approximately 40mW [80] (P_{link_ll}), resulting in over 5x power reduction. The different power assumptions are summarized in Table 3.4 and the power comparison for the different topologies are shown in Figure 3.16.

The comparison trend is very similar to the cost comparison that was shown earlier in Figure 3.12. The hypercube gives the highest power consumption while the conventional butterfly and the flattened butterfly give the lowest power consumption. For 1K node network, the flattened butterfly provides lower power consumption than the conventional butterfly since it takes advantage of the dedicated SerDes to drive

Component	**Power**
P_{switch}	40 W
P_{link_gg}	200 mW
P_{link_gl}	160 mW
P_{link_ll}	40 mW

Table 3.4: Power consumption of different components in a router.

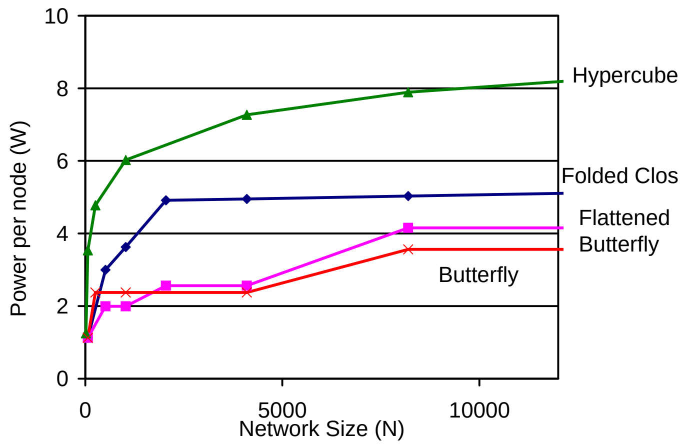


Figure 3.16: Power comparison of alternative topologies. The power consumption for the interconnection network normalized to N is plotted as the N is increased.

local links. For networks between 4K and 8K nodes, the flattened butterfly provides approximately 48% power reduction, compared to the folded-Clos because a flattened butterfly of this size requires only 2 dimensions while the folded-Clos requires 3 stages. However, for $N > 8K$, the flattened butterfly requires 3 dimensions and thus, the power reduction drops to approximately 20%.

3.6 Related Work

The flattened butterfly is similar to a generalized hypercube [10] (GHC) topology. The generalized hypercube is a k-ary n-cube network that uses a complete connection, rather than a ring, to connect the nodes in each dimension. In the 1980s, when this topology was proposed, limited pin bandwidth made the GHC topology prohibitively

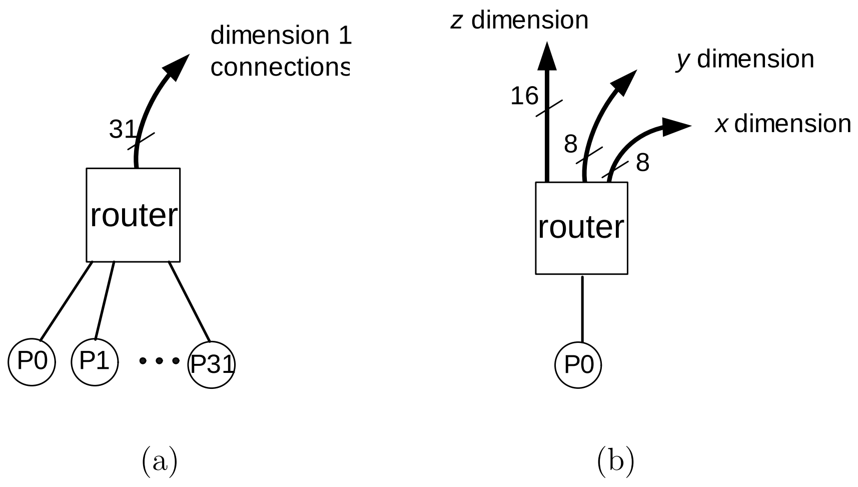


Figure 3.17: Block diagram of routers used in a 1K network for (a) flattened butterfly with one dimension and (b) (8,8,16) generalized hypercube.

expensive at large node counts. Also, without load-balancing global adaptive routing, the GHC has poor performance on adversarial traffic patterns.

The flattened butterfly improves on the GHC in two main ways. First, the flattened butterfly connects k terminals to each router while the GHC connects only a single terminal to each router. Adding this k-way concentration makes the flattened butterfly much more economical than the GHC - reducing its cost by a factor of k, improves its scalability, and makes it more suitable for implementation using high-radix routers. For example, routers used in a 1K node network of a flattened butterfly with one dimension and a (8,8,16) GHC are shown in Figure 3.17. With 32-terminal nodes per router, the terminal bandwidth of the flattened butterfly is matched to the inter-router bandwidth. In the GHC, on the other hand, there is a mismatch between the single terminal channel and 32 inter-router channels. If the inter-router channels are of the same bandwidth as the terminal channel, the network will be prohibitively expensive and utilization of the inter-router channels will be low. If the inter-router channels are sized at 1/32 the bandwidth of the terminal channel, serialization latency will make the latency of the network prohibitively high and the overall bandwidth of the router will be low, making poor use of the high-pin bandwidth of modern VLSI chips.

The second improvement over the GHC, which will be discussed in detail in Chapter 4, is the use of non-minimal globally-adaptive routing to load-balance adversarial traffic patterns and the use of adaptive-Clos routing with *sequential* allocators to reduce transient load imbalance. These modern routing algorithms enable the flattened butterfly to match the performance of a Clos network on adversarial traffic patterns. In contrast, a GHC using minimal routing is unable to balance load and hence suffers the same performance bottleneck as a conventional butterfly on adversarial traffic.

The Cray BlackWidow network [68] implements a high-radix modified folded-Clos topology using radix-64 routers. The network introduces *sidelinks* that connect neighboring subtrees together directly instead of using another stage of routers. A Rank1.5 BlackWidow network is similar to a one-dimensional flattened butterfly. However, the size of the routing table and packaging constraints limit the scalability of the Rank1.5 BlackWidow network to 288 nodes. Also, the BlackWidow router uses minimal routing and does not load balance across the sidelinks like the flattened butterfly with non-minimal routing. As the BlackWidow network scales to larger numbers of nodes, Rank2.5 and Rank3.5 networks are hybrids that resemble a flattened butterfly at the top level but with folded-Clos subnetworks at the lower levels. The flattened butterfly topology introduced in this paper extends the concept of sidelinks to create a topology where every link is a sidelink and there are no uplinks or downlinks to middle stage routers. Our work also improves upon the BlackWidow network by using global adaptive non-minimal routing to load-balance sidelinks.

The Flat Neighborhood Networks (FNN) [26] is an interconnection network proposed for clusters that is *flat* – i.e. there is only a single router between every pair of nodes. FNN partitions the terminal node bandwidth among the multiple routers and thus, provides a single intermediate hop between any two nodes. However, the network leads to an asymmetric topology and is very limited in its scalability.

To increase the path diversity of the butterfly network, alternative butterfly networks have been proposed. Additional stages can be inserted to the butterfly network [72] but adding additional stages to the butterfly ultimately leads to a Clos network. Dilated butterflies [47] can be created where the bandwidth of the channels in the butterflies are increased. However, as shown in Section 3.4.3, the network cost is dominated by the links and these methods significantly increase the cost of the network with additional links as well as routers.

3.7 Summary

Existing topologies such as the conventional butterfly or the folded-Clos (fat-tree) can exploit high-radix routers. However, conventional butterfly has limited path diversity which can lead to performance degradation while the folded-Clos provides high performance but incurs approximately a 2× increase in cost. In this chapter, we presented a topology called the *flattened butterfly* which approaches the path diversity of a folded-Clos while approaching the cost of a conventional butterfly. By eliminating the intermediate middle stages, the flattened butterfly provides a more cost-efficient topology compared to a folded-Clos. Through a detailed cost model, we showed the cost advantages of the flattened butterfly compared to alternative topologies. In the next chapter, we will discuss the impact of routing on high-radix networks and present the benefits of proper adaptive routing on such high-radix networks. We will also provide a performance comparison of the alternative high-radix topologies.

Chapter 4

Adaptive Routing in High-Radix Networks

In this chapter, we focus on routing in high-radix networks. Routing, which determines the path a packet takes from its source to its destination, can be classified as either adaptive routing, where the network state is used to determine the route, or oblivious routing, where the routing decision is made randomly or deterministically. Adaptive routing is often used as it provides higher performance. However, high-radix routers presents unique challenge in adaptive routing because of the large number of ports. Proper adaptive routing can reduce latency and provides less variance in the distribution of packet latency.

In Section 4.1, we first discuss the benefits of adaptive routing in a high-radix network by evaluating the folded-Clos topology, which provides high-path diversity where all paths are minimal. Even with all minimal paths, we show the benefits of adaptive routing, especially when nonuniformities in the network traffic exist with the presence of deterministic routing as well as faults in the network. Since the large number of ports in a high-radix router can impact the router cycle time, we also introduce randomization in the allocation algorithms to simplify the routing decision with minimal loss in performance. To reduce the implementation cost, we show how reduced precision simplifies the comparison logic and *precomputation* of the allocations minimizes the impact on the router pipeline delay.

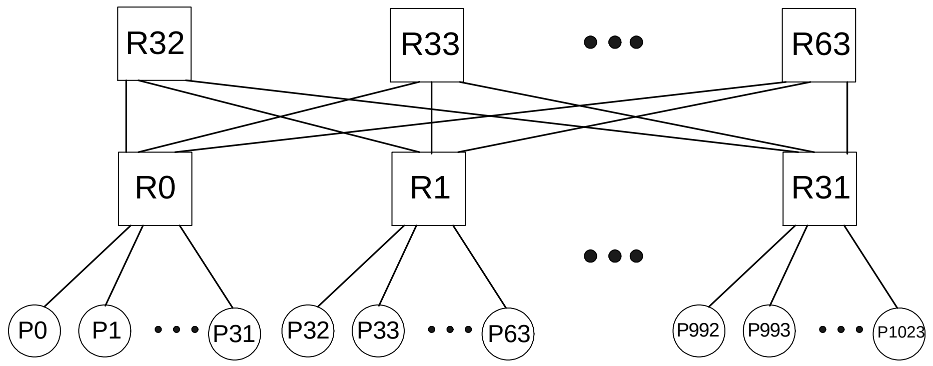


Figure 4.1: Block diagram of a 1K node high-radix folded-Clos network with radix-64 routers. P0-P1023 represents the terminals, R0-R31 represents the first level routers, and R32-R63 represents the second level routers.

In Section 4.2, we present the benefits of using global adaptive routing in the flattened butterfly topology which introduces non-minimal routing. We show how the use of non-minimal routing is critical to fully exploit the path diversity in the flattened butterfly and evaluate global adaptive routing against alternative routing algorithms.

4.1 Routing on High-Radix Folded-Clos Topology

4.1.1 Adaptive vs. Oblivious routing

In this section, we compare adaptive and oblivious routing on a high-radix folded-Clos network and discuss the benefits of adaptive routing. We show how adaptive routing is particularly beneficial with limited buffering and nonuniformities in the network traffic. We assume an ideal adaptive routing in this section using the *sequential* allocation algorithm. The different allocation algorithms will be discussed in Section 4.1.2.

Performance Evaluation

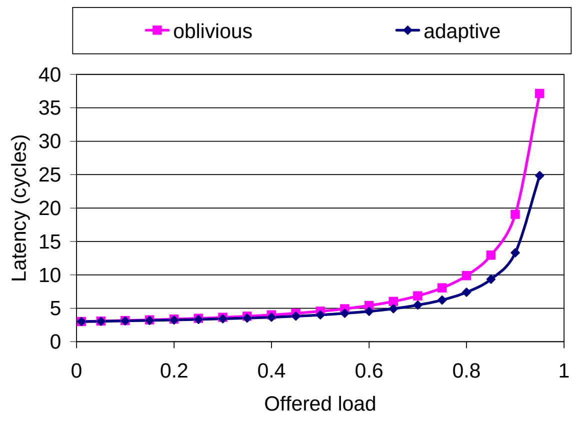
In this section, we provide additional simulation results to compare adaptive and oblivious routing. We use a cycle-accurate simulator to evaluate the performance of adaptive and oblivious routing in a folded-Clos network. We perform open-loop simulations [22] to evaluate the network and assume that the system has enough latency hiding so that the offered load is not affected by message latency.[1] We simulate a single-cycle input-queued router and the packets are injected using a Bernoulli process. The simulator is warmed up under load without taking measurements until steady-state is reached. Then a sample of injected packets are labeled during a measurement interval. The sample size was chosen such that the measurements are accurate to within 3% with 99% confidence. The simulation is run until all measurement packets are delivered.

As we will discuss in Chapter 5, the design of input-queued routers is problematic as the number of ports increase in high-radix routers. However, in order to generalize the results, we use input-queued routers and provide sufficient switch speedup so that the routers do not become the bottleneck in the network. We use radix-64 routers and create a 3-stage folded-Clos network with 1K nodes as shown in Figure 4.1. We evaluate the network performance using the worst-case uniform random (wc-UR) traffic pattern – each source sends traffic to a random destination whose common ancestor is the root of the network – as well as permutation traffic patterns. Credit-based flow control is used between routers to maintain the buffer information of downstream routers. Since we are focusing on the routing of the network, we assume only a single virtual channel [23] and use a packet size of 1 flit. Longer packet sizes generally follow the same trend in the comparison but when necessary to clarify some of the results, we vary the packet size.

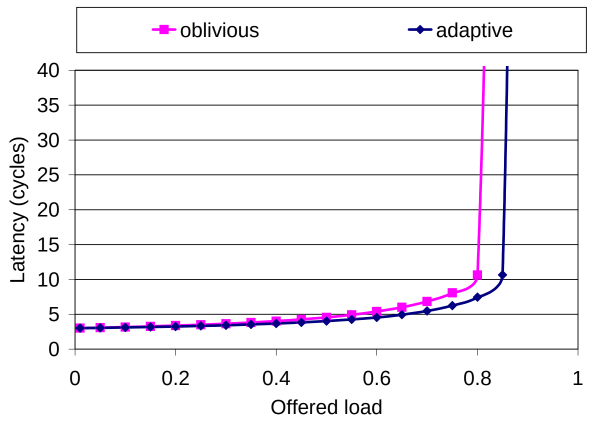
Infinite Buffers

The simulation results comparing adaptive and oblivious routing with infinite buffers is shown in Figure 4.2(a). The throughput is identical as both routing algorithms achieve nearly 100% throughput with adaptive routing resulting in lower latency at higher loads. For oblivious routing, the middle stages are chosen randomly – thus, with sufficient buffering, the *expected* or the average load across all the outputs will

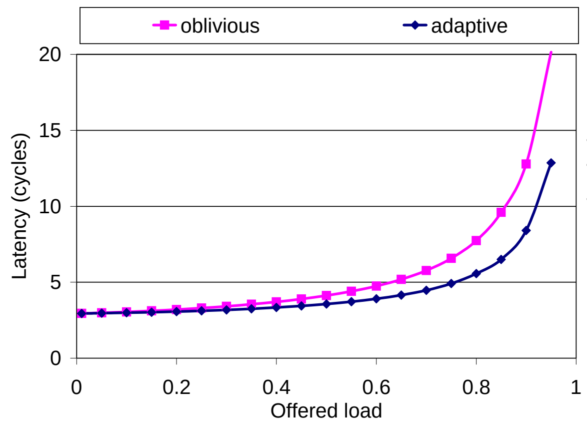
[1] As an example, the processors in the BlackWidow system [68] can support thousands of outstanding global memory references and provide latency hiding.



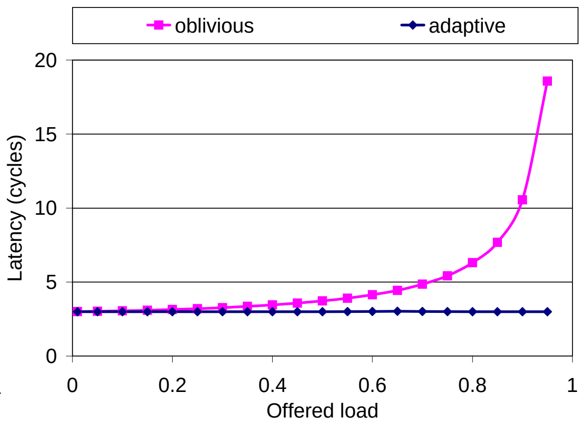
(a)



(b)

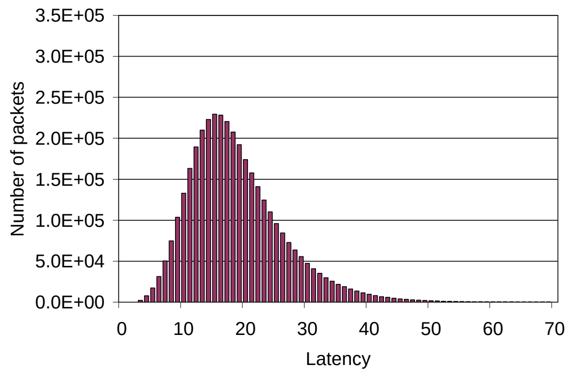


(c)

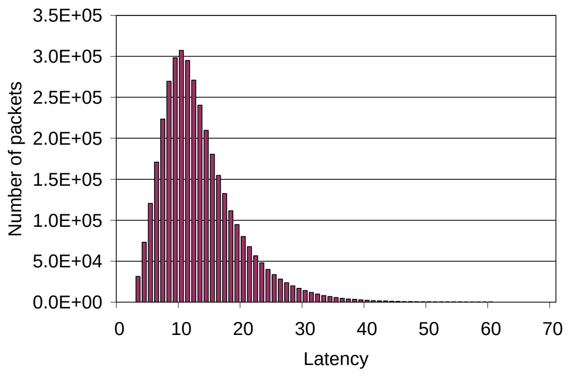


(d)

Figure 4.2: Adaptive and oblivious routing comparison of the latency vs. offered load in the folded-Clos network with (a) wc-UR traffic with infinite buffers (b) wc-UR traffic with 16 buffers (c) bit reverse traffic pattern and (d) bit complement traffic pattern.



(a)



(b)

Figure 4.3: Latency distribution of packets with an offered load of 0.9 with (a) oblivious routing and (b) adaptive routing.

be the same. Thus, oblivious routing results in the same throughput as adaptive routing with infinite buffers.

However, adaptive routing provides lower latency at higher offered loads since the routing decisions are made adaptively each cycle to load balance across the uplinks. To illustrate the benefit of adaptive routing, we plot the distribution of packet latency in Figure 4.3 for oblivious and adaptive routing at an offered load of 0.9. The average latency of oblivious routing is approximately 38% higher than adaptive. In addition, adaptive routing has a tighter latency distribution with a standard deviation that is 20% less than that of oblivious routing. As a result, the poor instantaneous decisions of oblivious routing lead to higher average latency and a larger latency distribution of the packets compared to adaptive routing. Reducing the variance in the packet latency is significant as it will improve the global synchronization time across the whole system.

Other traffic patterns such as bit reverse permutation [22] follows the same trend as the wc-UR traffic pattern (Figure 4.2(c)). With a congestion-free traffic pattern [33] such as the bit complement permutation, adaptive routing results in a constant delay regardless of the offered load [7] while congestion increases latency at higher offered load with oblivious routing (Figure 4.2(d)).

Finite Buffers

When buffering is limited, the throughput of oblivious routing suffers compared to adaptive routing as shown in Figure 4.2(b) when the input buffers are limited to 16 entries.[2] Adaptive routing provides approximately 10% higher throughput as oblivious routing does not consider the state of network – i.e. the randomly selected output maybe not be available because of lack of buffer space.

The poor instantaneous load-balancing of oblivious routing can be shown by a *snapshot* of the buffer utilization of the middle stage routers during simulation. For each middle stage router, we plot the size of the input buffer with the highest occupancy in Figure 4.4 at an offered load of 0.8. The average queue utilization is under 1 for both routing algorithms (0.78 for oblivious and 0.33 for adaptive) but the distribution of the maximum queue occupancy is different. With oblivious routing, some of the buffers are filled to capacity (16 entries) and average of the maximum buffer depth is approximately 5 entries. In contrast, the average of the maximum buffer depth is only 3 entries with adaptive routing and the maximum value is only 7, or roughly 50% of the capacity. Because of this imbalance in buffer utilization, oblivious routing results in not only higher latency but also lower throughput with limited buffering.

Nonuniformity - Presence of Deterministic Traffic

Additional benefits of adaptive routing can be observed in the presence of nonuniformity. Because of the symmetry in the topology, nonuniformity can not be created by the traffic pattern itself since the traffic can be distributed across all the middle stages.

The nonuniformity can result from deterministic routing being used in the network. In a shared-memory multiprocessor, it is often necessary to ensure ordering of requests to a given cache-line memory address because of the memory consistency model. Therefore, deterministic routing such as that used by the Cray BlackWidow

[2]16 buffer entries are sufficient to cover the credit latency.

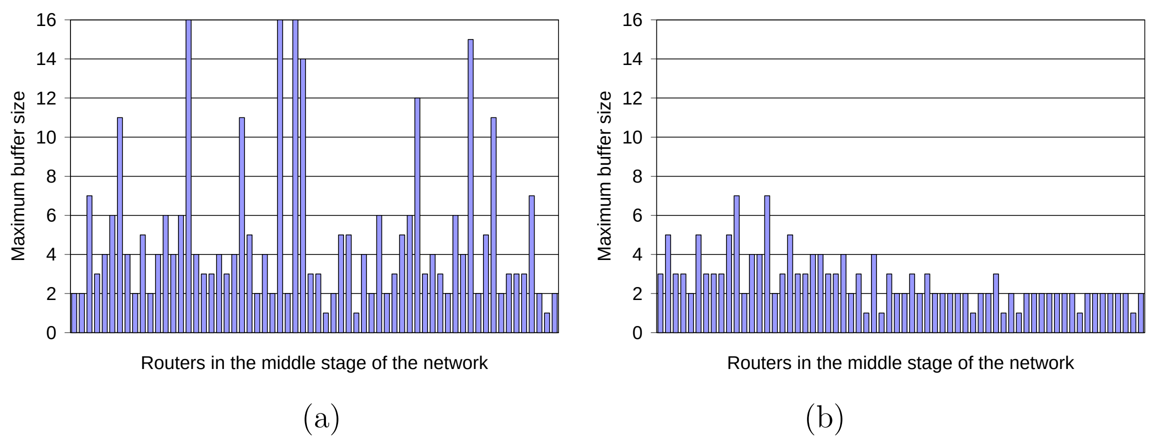
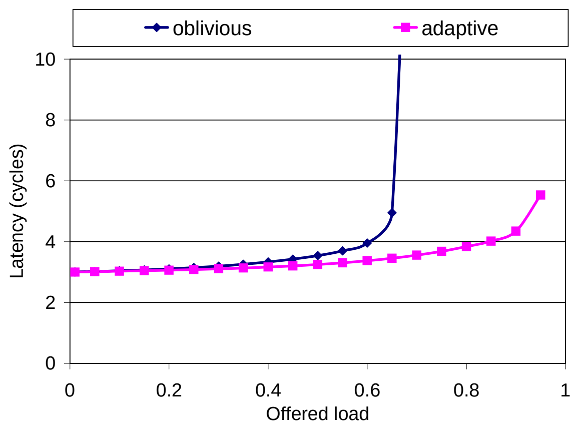


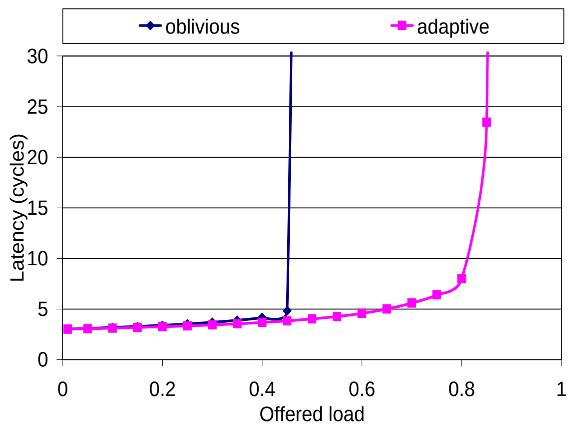
Figure 4.4: A *snapshot* of the maximum buffer size of the middle stage routers in a 1K node folded-Clos network. The distribution is shown for (a) oblivious and (b) adaptive routing at an offered load of 0.8 and the buffer depth of the routers are 16 entries.

network[68], can be used to provide in-order delivery of all request packets at a cache-line address granularity for a source-destination pair. Since response packets in the network do not require ordering they can be routed using either oblivious or adaptive routing. By taking into account the congestive state of the network, adaptive routing avoids any nonuniformities that may be introduced as a result of deterministic routing of the request traffic. Oblivious routing does not take into account these potential nonuniformities and may randomly select an output port which has a substantial amount of deterministic request traffic en route. In effect, the adaptive routing will "smooth out" any nonuniformities in the traffic pattern to load balance the set of available links.

To illustrate this nonuniformity, we simulate a traffic pattern where each node i sends traffic to $(i + k)\ mod\ N$ where k is the radix of the routers and N is the total number of nodes. Using this traffic pattern, 50% of the traffic is routed using deterministic routing and the remaining 50% of the traffic is routed either adaptively or obliviously. The simulation result is shown in Figure 4.5(a). With oblivious routing, the throughput is limited to under 70% but with adaptive routing, 100% throughput can be achieved. The deterministically routed traffic cause nonuniform channel loads in the various middle stages since deterministic routing does not exploit the path



(a)



(b)

Figure 4.5: Routing comparison with nonuniformity in the traffic pattern. The latency vs. offered load comparison of adaptive and oblivious routing in the folded-Clos network is shown for when (a) half of the traffic is routed using deterministic routing and (b) network with faults.

diversity and is routed through the same middle stage. Adaptive routing allows the other traffic to be routed around these nonuniform loads while oblivious routing can not avoid the nonuniformity, leading to lower throughput.

Nonuniformity - Faults in the Network

Nonuniformity can also be created by oblivious routing in the presence of faults in the network. An example is shown in Figure 4.6. Assume that the downlink from R4→R0 is faulty[3] and we observe the wc-UR traffic generated from nodes connected to R1. Any traffic generated for R0 can not be routed through R4 since they can not reach its destination, resulting in the traffic being load balanced across the other three routers (R5-R7). However, traffic destined for R2 and R3 will be equally distributed across all four uplinks. As a result, the uplink from R1→R4 will be underutilized while the other three uplinks will be over utilized – limiting the throughput of the network. To load balance appropriately, the traffic for R2 and R3 should utilize the R1→R4 uplink more so that the traffic is balanced.

[3]The corresponding uplink will also need to be disabled.

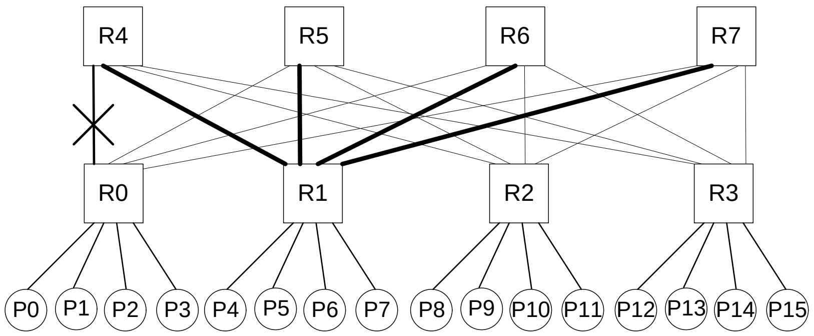


Figure 4.6: Block diagram of a radix-8 3-stage folded-Clos network with a fault. By using oblivious routing, the uplinks to the middle stages are not load balanced.

To evaluate the impact of faults on adaptive and oblivious routing, we simulate a faulty network with the 1K network shown in Figure 4.1 with approximately 1.5% of the links between stage1 and stage2 routers (16 of the 1024 links) assumed to be faulty. The simulation results on this network with the wc-UR traffic pattern is shown in Figure 4.5(b). By load balancing appropriately across the healthy links, adaptive routing leads to approximately 2× improvement in throughput.[4]

4.1.2 Allocation Algorithms in Adaptive Routing

Adaptive routing in a folded-Clos requires an *allocation* algorithm since the outputs (middle stages) need to be appropriately assigned to the inputs. We discuss the different allocation algorithms and compare their performances in this section.

Algorithm Description

To load balance in a folded-Clos network, adaptive routing selects the middle stage with the least amount of congestion – i.e. middle stage with the largest amount of buffering available. Since simulation is done with input-queued routers, *credits* from the downstream routers are used to load-balance. Thus, middle stages with more credits (more buffer space) is preferred over those with fewer credits.

[4]In the simulation setup, we assumed 16 of the 32 links connected to R32 and R33 of Figure 4.1 are faulty. Different simulation setup will result in different amount of benefit using adaptive routing.

We evaluate the following four allocation algorithms. Unless stated otherwise, ties (equal credit counts) are broken randomly.

- *sequential* : Each input i makes its adaptive decision after inputs 0 through $i-1$ have made their decisions and updated the state of the network. The allocation algorithm assigns outputs to each input one at a time, taking into account the previous allocations of this cycle. To provide fairness, starting input is selected randomly each cycle.
- *greedy* : Each input adaptively selects an output independently. As a result, each input does not take into account the routing decisions made by the other inputs in the same cycle.
- *sequential_r*(n) : A randomized version of *sequential* with n samples. Each input i selects n random outputs and adaptively selects among the n random outputs after inputs 0 through $i-1$ have made their routing decision. When $n = 1$, *sequntial_r*(1) is similar to oblivious routing and *sequntial_r*(k) is similar to the *sequential* algorithm where k is the radix of the router.
- *greedy_r*(n) : A randomized version of *greedy* with n samples. Each input selects n random outputs and adaptively selects among them. *greedy_r*(k) is identical to *greedy* and *greedy_r*(1) is identical to oblivious routing.

In the *sequential* and the *sequential_r*(n) allocation algorithms, if multiple outputs have the same credit count, the ties are broken randomly. However, if one of the outputs with the same credit count has already been selected by a previous input in the same cycle, we provide priority to the other outputs. For example, if four outputs $\{O_0,O_1,O_2,O_3\}$ have a credit count $\{2,2,2,3\}$, the first input (I_0) would select O_3. The credit count for O_3 would be decremented by 1, resulting in a credit count of $\{2,2,2,2\}$. For the next input (I_1), all of the credits are equal and it can select any output to load balance. However, since O_3 has been selected by another input, we provide preference to the other three outputs (O_0,O_1,O_2). Using this policy to break ties simplifies the switch scheduling by not overloading an output and reduces congestion.

Algorithm Comparison

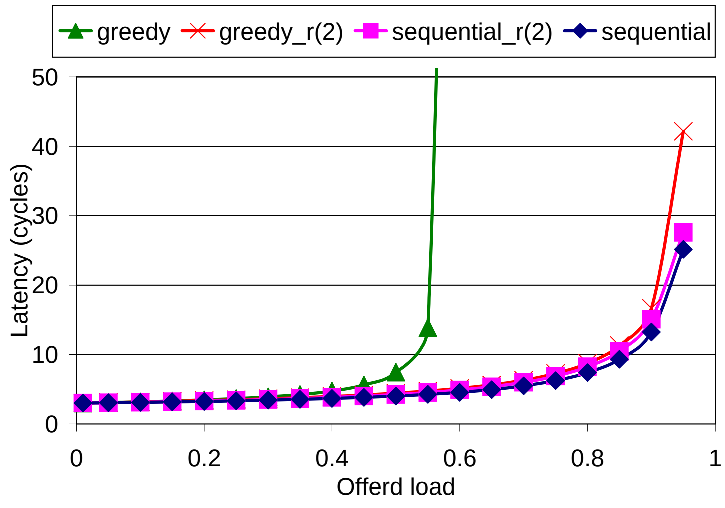
We simulate the different allocation algorithms using the simulation setup described in Section 4.1.1. For the randomized allocation algorithms, we use $n = 2$. The allocation algorithms are compared in Figure 4.7 using the wc-UR traffic pattern. The results for other traffic patterns follow the same trend.

With unlimited buffering, *sequential* provides the best performance as it leads to the lowest latency (Figure 4.7(a)). The *sequential_r*(2) performs comparable to *sequential* but leads to slightly higher latency near saturation (approximately 10% higher at an offered load of 0.95). The *greedy_r*(2) also provides the same throughput but leads to higher latency, approximately 60% higher at an offered load of 0.95.

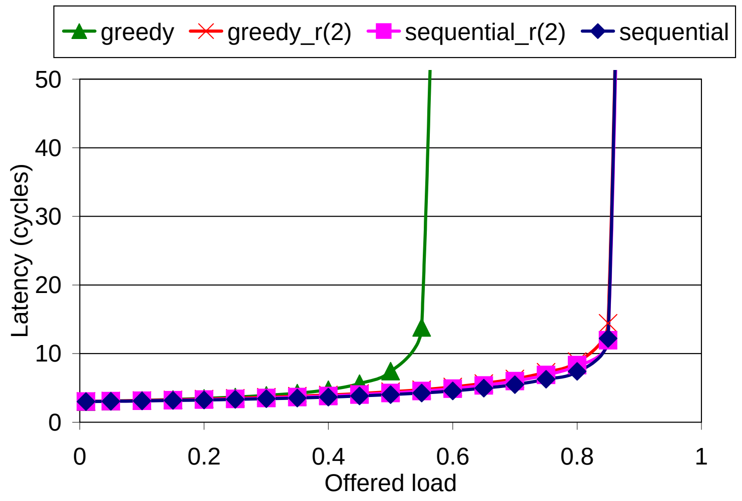
The difference in latency between *greedy_r*(2) and *sequential* is minimized with limited buffering (Figure 4.7(b)). With unlimited buffers, *greedy_r*(2) might randomly select a *bad* output – i.e. an output which has a lot of packets. However, with limited buffering, if bad outputs are selected, the packet can be stalled when the buffers are full and allocation is re-attempted in the next cycle to avoid the *bad* outputs. Thus, the latency difference is less than 20% at an offered load of 0.85 near saturation.

Regardless of the amount of buffering, the *greedy* algorithm performs poorly and saturates at less than 60%. With the *greedy* algorithm, each input makes its routing decision independent of the other inputs. Thus, the routing decision might be an optimal *local* decision but could be a poor *global* decision in attempting to load balance. To illustrate this behavior, we use the same example from Section 4.1.2. If the credit count was {2,2,2,3} for the four outputs {O_0,O_1,O_2,O_3} and if a new packet arrives at all four inputs, all of the inputs would select O_3. This allocation would be an optimal local decision for the four inputs but does not globally load balance across all the outputs and leads to a poor allocation. As a result, congestion is created at an output and the poor allocation decision creates a head-of-line blocking effect [39] and limits the throughput of the router.

It is worth noting that the *greedy* algorithm is not problematic with low-radix routers but becomes problematic with high-radix routers. The throughput of the *greedy* algorithm is compared in Figure 4.8(a) on a 4K network as the radix of the



(a)



(b)

Figure 4.7: Adaptive routing comparisons with (a) infinite buffers and (b) 16 buffers using wc-UR traffic pattern.

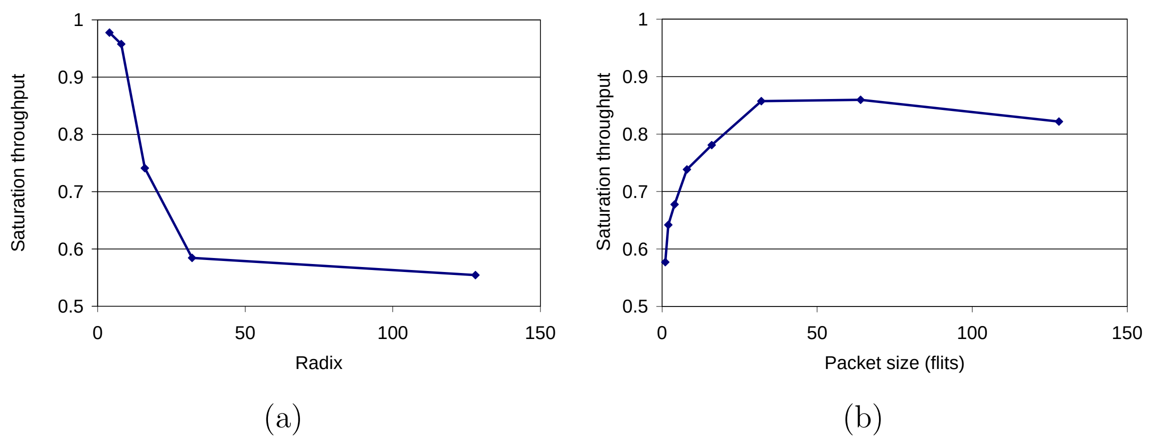


Figure 4.8: The impact on the saturation throughput as (a) radix and (b) packet size is varied using the *greedy* routing algorithm. Higher radix and smaller packet size limits the throughput of the *greedy* algorithm.

routers is varied between radix-4 and radix-128 with a single flit packet. Lower radix networks achieve almost 100% throughput but the throughput drops to under 80% with radix-16 and beyond radix-32, the throughput is under 60%. The packet size also impacts the performance of *greedy* algorithm as large packet size increases throughput (Figure 4.8(b)). Since the routing decision is made only for the head flit of a packet, increasing packet size decreases the probability of having 2 or more new packets arriving in the same cycle – reducing the chance of output collision from the poor routing decision. In Figure 4.9, we plot this probability as a function of offered load. The probability approaches 1 very quickly for radix-64 router which explains the poor performance of high-radix routers for the *greedy* algorithm using 1-flit packets. However, the probability gradually approaches 1 for radix-8 router, With a packet size of 8-flits in a radix-64 router, the probability is reduced and behaves very similar to a radix-8 router with 1-flit packets. Thus, it is the ratio between the radix and the packet length that determines the performance of *greedy* algorithm.

To evaluate the impact of the parameter n, we vary n in the randomized allocation algorithms ($sequential\_r(n)$ and $greedy\_r(n)$) and plot the latency at an offered load of 0.9 in Figure 4.10. When $n = 1$, the randomized allocation algorithms are identical to oblivious routing since only 1 randomly selected output is used. As n increases from

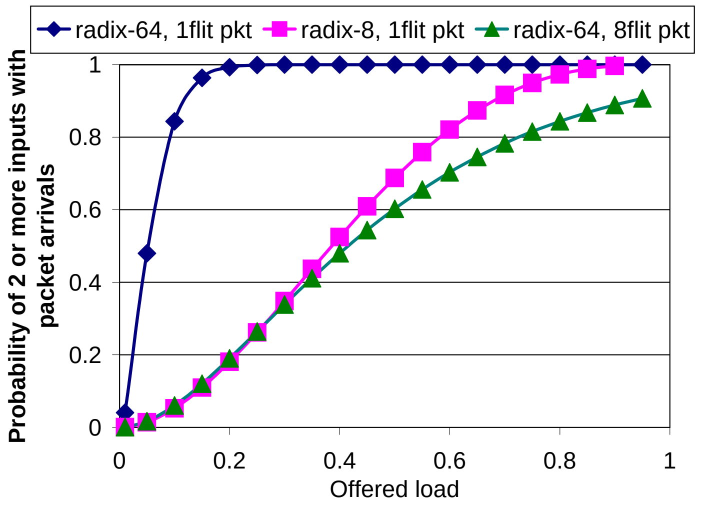


Figure 4.9: Probability of two or more packets arriving in the same cycle as the radix and the packet size is varied.

1 to 2, there is approximately 10% reduction in latency for *greedy_r*(n). However, as n is increased further, the latency *increases* significantly and is beyond the scale of the plot. As n approaches 32, the allocation algorithm behaves like *greedy* and the network is no longer stable as the offered load of 0.9 exceeds the throughput.[5] Similar to *greedy_r*(1), *sequential_r*(1) behaves identical to oblivious routing. By increase n from 1 to 2, there is over 20% reduction in latency with *sequential_r*(1). However, increasing n from 2 to 32 results in less than 10% reduction in latency. Thus, most of the performance gain can be achieved by using only two samples.

The randomized algorithm were implemented assuming that the random choices are not necessarily unique - e.g. for *sequential_r*(2), the two randomly selected outputs can be the same output. Simulations show that the uniqueness of the random choices has minimal impact on the latency of the allocation algorithm. As a result, *sequential_r*(32) latency is slightly higher than the *sequential* but by less than 1%.

[5]Although radix-64 routers is used in the simulation, only 32 output selections are possible in routing upstream in the folded-Clos. Thus, the maximum size of n is 32.

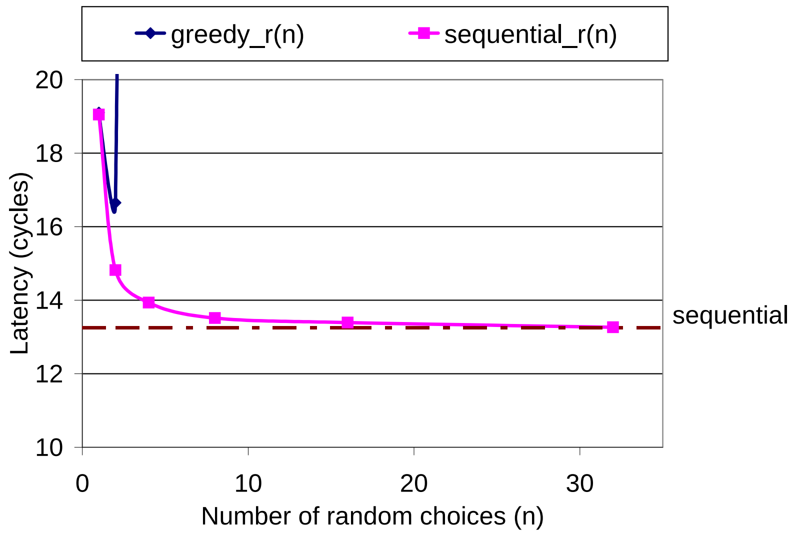


Figure 4.10: Randomized adaptive allocation algorithm comparison as n is varied for *sequential_r*(n) and *greedy_r*(n). The lower bound of the algorithm is shown by the *sequential* line.

4.1.3 Cost Analysis

Although adaptive routing provides performance benefits, the cost of implementation complexity, in terms of router latency and area, needs to be considered. For deterministic routing or source routing where only bit manipulation is required or oblivious routing where only a random number needs to be generated, the routing pipeline delay will be minimal. However, earlier work showed that introducing adaptive routing can increase the complexity and the cycle time of a router [14]. The larger number of ports in a high-radix router can further increase the routing complexity. For example, the Cray YARC router requires 4 clock cycles for the routing decision [68].

A timeline of adaptive routing in a high-radix network is shown in Figure 4.11. The three main components to the latency of adaptive routing in high-radix routers are the following:

1. Collecting the network state (e.g. availability of the outputs and the output credit information) as well as *requests* from each input.

2. Allocation based on the network state and the *requests*.

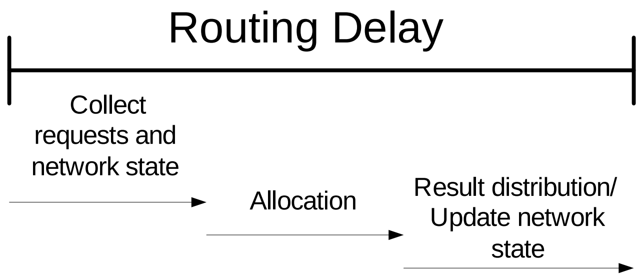

Routing Delay
Collect requests and network state
Allocation
Result distribution/ Update network state

Figure 4.11: Timeline of delay in adaptive routing in a high-radix folded-Clos network.

3. Updating the network state and distributing the outputs assigned to each input.

In this section, we provide a qualitative comparison of the different adaptive allocation algorithms discussed in Section 4.1.2. Then, we evaluate two different techniques that reduce the complexity of adaptive routing in a high-radix folded-Clos network. By using imprecise queue information, the complexity of the route allocation can be reduced. The precomputation of the allocations can effectively hide the router latency of adaptive routing. We evaluate their impact on performance and show that there is minimal performance loss. The use of imprecision can be used for all four of the algorithms but the precomputation can only be used for *greedy* and *greedy_r*(n) algorithm since they are distributed algorithms.

Algorithm Cost Comparison

Among the adaptive algorithms discussed in Section 4.1.2, *sequential* requires a centralized routing structure that collects all of the requests, performs the allocation, and distributes the results. The complexity of such routing structure grows as $O(k^2)$ and becomes prohibitively expensive to implement. The *sequential_r*(n) routing algorithm, even with $n = 2$, still requires a central routing structure since each input is routed sequentially.

The *greedy* algorithm does not require a centralized structure as the routing logic can be duplicated at each input and be distributed. However, the routing algorithm still requires the distribution of the output information to all of the inputs. In addition, the comparison logic at each input needs to compare all the outputs,

number of bits	Description
0	no buffer depth information used – only whether an output is available or not
1	only the MSB is used
2	two MSBs is used
3	three MSBs is used
4	full buffer information is used

Table 4.1: Different values of precisions used to evaluate the impact of precision in adaptive routing. Buffer depth of 16 entries is assumed.

requiring a significant amount of logic. The $greedy\_r(n)$ algorithm can simplify the implementation as only n comparisons need to be made. Simulation earlier showed that $n = 2$ performs well – thus, only a single comparison is needed.

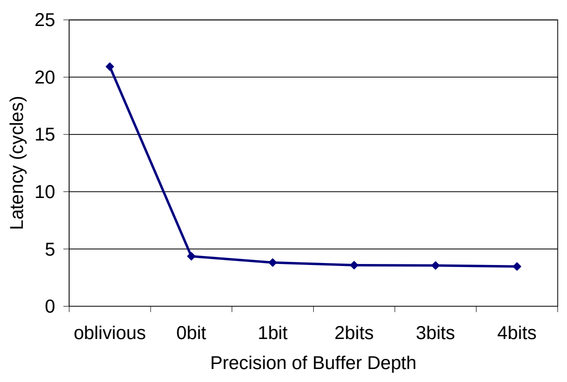
Precision in Adaptive Routing

The buffer depth (credits) is used to load-balance properly in adaptive routing and results so far assumed that the *full* information was available – i.e. the exact buffer depth was available from the credits. For a buffer with q entries, $log_2 q$ bits are needed to obtain the exact buffer information. However, $log_2 q$ bit comparators can be costly with the larger number of ports in a high-radix router. As a result, the YARC router only uses 1 or 2 bits of information to make its adaptive decision [68].[6]

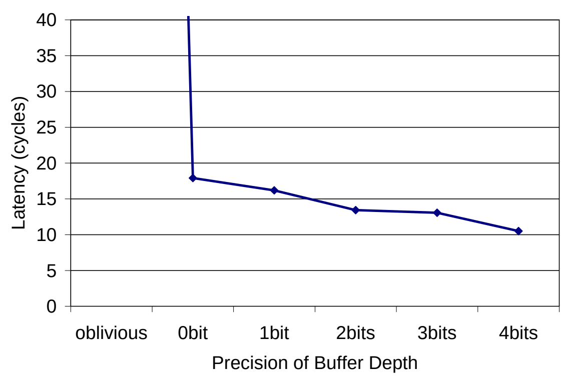
Table 4.1 describes the different values of precisions that are used to evaluate the impact of the precision. We assume a buffer depth of 16 and use the *sequential* algorithm in our evaluation. Using 0 bits of information corresponds to an allocation which does not consider the queue depth but considers only whether an output is available or not.[7] An output is not available if another input with a multi-flit packet is transmitting to the output or if the downstream buffer is full. By using only the most

[6]The routing decision at the input buffers of YARC use 1 bit to select the row buffer and 2 bits are used to select the output within the column.

[7]This adaptive allocation is similar to the adaptive routing used in CM-5 [53].



(a)



(b)

Figure 4.12: Latency comparison near saturation for network simulations with wc-UR traffic using (a) 1 flit packets and (b) 10 flit packets as the precision of allocation is varied . The network shown in Figure 4.1 was used for the simulations.

significant bit (MSB), the adaptive information will be used to differentiate whether the buffer has more or less than 8 entries. By using 2 bits, the buffer information is defined in granularity of 4 entries – e.g. 00 corresponds to less than 4 entries, 01 corresponds to 4 or more entries but less than 8 entries, and so forth. Using 3 bits results in a finer granularity and 4 bits allows the exact queue information to be used.

We plot the latency near saturation throughput as the precision is varied in Figure 4.12. With single flit packets, there is a significant difference between oblivious and adaptive routing but only a small difference between any of the different precision schemes – the difference between 0 bits and 4 bits is less than 10% (Figure 4.12(a)). With longer packets, the precision has more significant impact on the latency as 4 bits of precision can reduce the latency by over 60%, compared to using 0 bits (Figure 4.12(b)). However, by using only 2 or 3 bits of precision, the latency can still be reduced by over 40% compared to using 0 bits.

For the uniform traffic patterns, the throughput is very similar regardless of the precision used and difference in latency near saturation was compared. However, with the nonuniformity such as the one discussed in Section 4.1.1, the different precision results in different throughput as shown in Figure 4.13. Using 0 bit of information still outperforms oblivious routing (see Figure 4.5(b)) but results in approximately

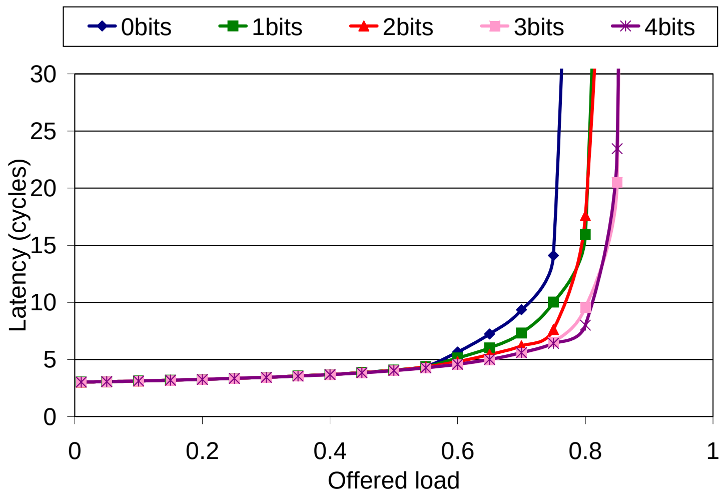


Figure 4.13: Impact of precision with nonuniform traffic pattern.

15% reduction in throughput, compared to using all 4 bits. By using only 2 bits of precision, the difference can be reduce by half and 3 bits of precision performs nearly identical to using all 4 bits of precision.

Precomputation

To reduce the impact of adaptive routing on the router latency, the allocation can be *precomputed*. By using the queue information in the previous cycle, the routing decision can be precomputed and be available when a new packet arrives. The precomputation of allocations can be utilized with minimal loss in performance for the following reasons.

- The queue depth will change minimally from cycle to cycle.
- When full precision is not used, the change in the credit will have minimal impact – e.g. if only the 2 bits are used for adaptive decisions, the change in the lower 2 bits will have minimal impact.
- With the use of randomization, even if some of the data is *stale*, it might not impact the results.

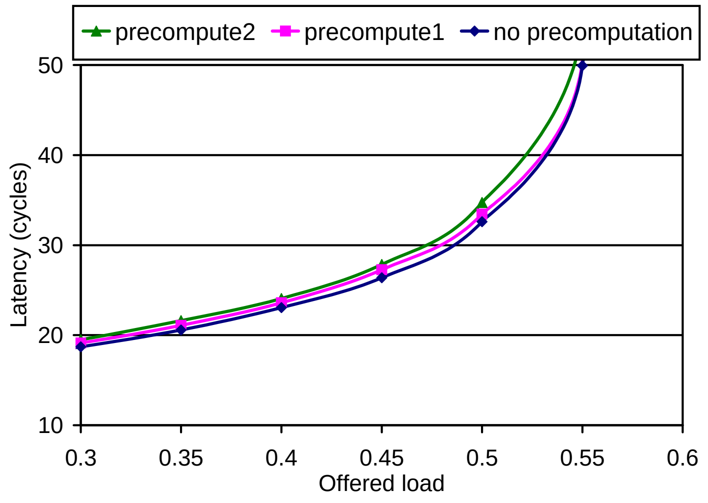


Figure 4.14: Performance comparison with the use of precomputation.

The performance comparison with precomputation is shown in Figure 4.14 using $greedy\_r(2)$ algorithm. We compare the performance without precomputation to precompute1, where the output is calculated in the previous cycle, and precompute2 where the output is calculated in 2 cycle advance. The comparison is shown for the nonuniform traffic from Section 4.1.1 with a 10 flit packets using only 2 bits of precision. Both precompute1 and precompute2 perform comparable to no precomputation, with precompute2 resulting in approximately 10% higher latency near saturation. With minimal loss in performance, precompute2 will allow an extra cycle to distribute the routing information – further minimizing the impact of wires and reduce the router pipeline delay as the routing results can be computed in advance.

4.2 Routing on Flattened Butterfly Topology

In this section, we describe routing algorithms for the flattened butterfly and compare their performance on different traffic patterns. We also compare the performance of the flattened butterfly to alternative topologies.

4.2.1 Routing Algorithm Descriptions

We evaluate the flattened butterfly on five routing algorithms: minimal adaptive (MIN AD), Valiant's non-minimal oblivious algorithm (VAL), the UGAL non-minimal adaptive algorithm (UGAL), a variant of UGAL using sequential allocation (UGAL-S), and non-minimal adaptive routing in a flattened Clos (CLOS AD). We describe each briefly here. We consider routing in a k-ary n-flat where the source node s, destination node d, and current node c are represented as n-digit radix-k numbers, e.g., $s_{n-1}, \ldots, s_0$. At a given step of the route, a channel is productive if it is part of a minimal route; that is, a channel in dimension j is productive if $c_j \neq d_j$ before traversing the channel, and $c_j = d_j$ after traversing the channel.

We assume an input-queued virtual channel router [22]. Adaptive algorithms estimate output channel queue length using the credit count for output virtual channels which reflects the occupancy of the input queue on the far end of the channel. For MIN AD and UGAL, the router uses a *greedy* allocator: during a given routing cycle, all inputs make their routing decisions in parallel and then, the queuing state is updated en mass. For UGAL-S and CLOS AD, the router uses a *sequential* allocator where each input makes its routing decision in sequence and updates the queuing state before the next input makes its decision.

Minimal Adaptive (MIN AD): The minimal adaptive algorithm operates by choosing for the next hop the productive channel with the shortest queue. To prevent deadlock, n' virtual channels (VCs) [23] are used with the VC channel selected based on the number of hops remaining to the destination.

Valiant (VAL) [77]: Valiant's algorithm load balances traffic by converting any traffic pattern into two phases of random traffic. It operates by picking a random intermediate node b, routing minimally from s to b, and then routing minimally from b to d. Routing through b perfectly balances load (on average) but at the cost of doubling the worst-case hop count, from n' to $2n'$ and destroying any locality. While any minimal algorithm can be used for each phase, our evaluation uses dimension order routing. Two VCs, one for each phase, are needed to avoid deadlock with this algorithm.

Universal Globally-Adaptive Load-balanced (UGAL [73], UGAL-S) : UGAL chooses between MIN AD and VAL on a packet-by-packet basis to minimize the estimated delay for each packet. The product of queue length and hop count is used as an estimate of delay. With UGAL, traffic is routed minimally on benign traffic patterns and at low loads, matching the performance of MIN AD, and non-minimally on adversarial patterns at high loads, matching the performance of VAL. UGAL-S is identical to UGAL but with a sequential allocator.

Adaptive Clos (CLOS AD): Like UGAL, the router chooses between minimal and non-minimal on a packet-by-packet basis using queue lengths to estimate delays. If the router chooses to route a packet non-minimally, however, the packet is routed as if it were adaptively routing to the middle stage of a Clos network. A non-minimal packet arrives at the intermediate node b by traversing each dimension using the channel with the shortest queue for that dimension (including a “dummy queue” for staying at the current coordinate in that dimension). Like UGAL-S, CLOS AD uses a sequential allocator. The routing is identical to adaptive routing in a folded-Clos where the folded-Clos is flattened into the routers of the flattened butterfly. Thus, the intermediate node is chosen from the closest common ancestors and not among all nodes. As a result, even though CLOS AD is non-minimal routing, the hop count is always equal or less than that of a corresponding folded-Clos network.

4.2.2 Evaluation & Analysis

We use cycle accurate simulations to evaluate the performance of the different routing algorithms in the flattened butterfly. We simulate a single-cycle input-queued router switch with sufficient switch speedup so that the routers do not become the bottleneck in the network and use the simulation setup described earlier in Section 4.1.1. We simulate a 32-ary 2-flat flattened butterfly topology ($k' = 63$, $n' = 1$, $N = 1024$). Simulations of other size networks and higher dimension flattened butterfly follow the same trend and are not presented due to space constraints. We simulate single-flit packets[8] and assume the total buffering per port is 32 flit entries. Although input-queued routers have been shown to be problematic in high-radix routers[45], we use

[8]Different packet sizes do not impact the comparison results in this section.

input-queued routers but provide sufficient switch speedup in order to generalize the results and ensure that routers do not become the bottleneck of the network.

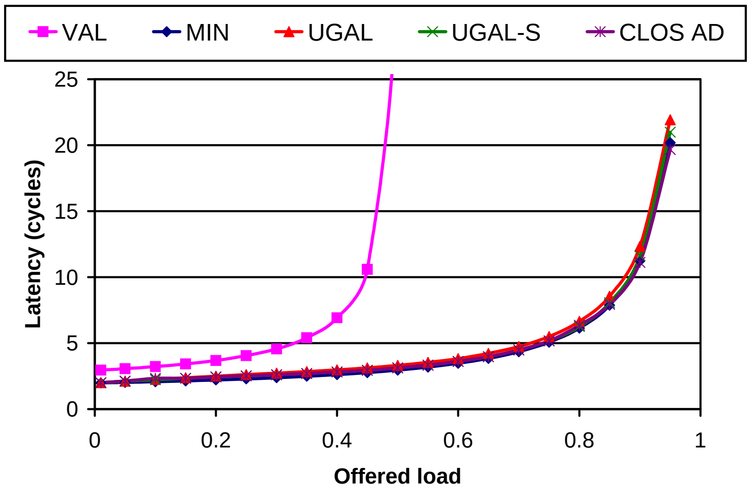
We evaluate the different routing algorithms on the flattened butterfly using both benign and adversarial traffic patterns. Uniform random (UR) traffic is a benign traffic pattern that balances load across the network links. Thus, minimal routing is sufficient for such traffic patterns and all of the routing algorithms except VAL achieve 100% throughput (Figure 4.15(a)). VAL achieves only half of network capacity regardless of the traffic pattern[9][74]. In addition, VAL adds additional zero-load latency with the extra hop count associated with the intermediate node.

We also simulate an adversarial traffic pattern where each node associated with router R_i sends traffic to a randomly selected node associated with router R_{i+1}. With this traffic pattern, all of the nodes connected to a router will attempt to use the same inter-router channel. Non-minimal routing is required to load balance this traffic pattern by spreading the bulk of the traffic across the other inter-router channels. Figure 4.15(b) compares the performance of the routing algorithms on this pattern. MIN is limited to 1/32 or approximately 3% throughput while all of the non-minimal algorithms achieve 50% throughput (the maximum possible on this traffic).

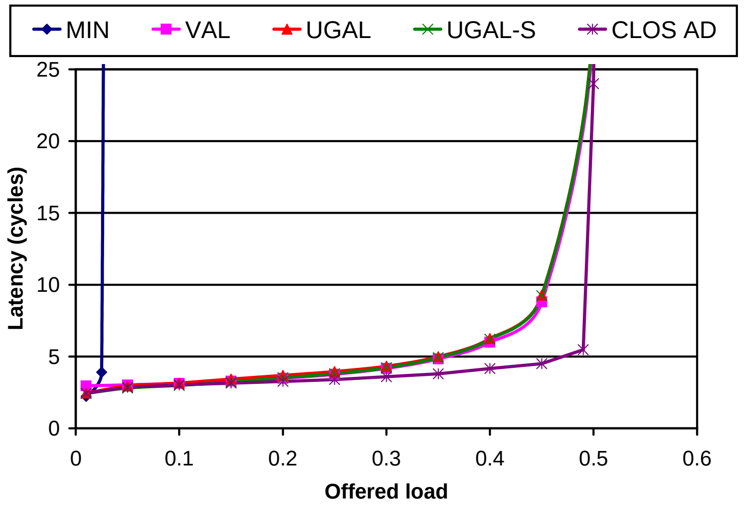
With the adversarial traffic pattern, CLOS AD provides much lower latency near saturation — nearly half the latency of UGAL-S at an offered load of 0.45. This reduction in latency is analogous to the reduction in latency achieved by adaptive routing compared to oblivious routing in a Clos network as shown in Section 4.1. Because CLOS AD adaptively picks the intermediate node, it is able to dynamically load balance traffic across the intermediate nodes and links. VAL, UGAL, and UGAL-S obliviously pick the intermediate node which balances average traffic across nodes and links but results in transient load imbalance that increases latency.

To highlight the effects of transient load imbalance, Figure 4.16 shows the time required by each algorithm to deliver a batch of traffic normalized to batch size. We use the adversarial traffic pattern described earlier. As the batch size increases, the normalized latency approaches the inverse of the routing algorithm throughput. For

[9]The capacity of the network (or the ideal throughput for UR traffic) is given by $2B/N$ where B is the total bisection bandwidth and N is the total number of nodes [22]. For the flattened butterfly, similar to the butterfly network, $B = N/2$, thus the capacity of the flattened butterfly is 1.



(a)



(b)

Figure 4.15: Routing algorithm comparisons on the flattened butterfly with (a) uniform random traffic and (b) worst case traffic pattern.

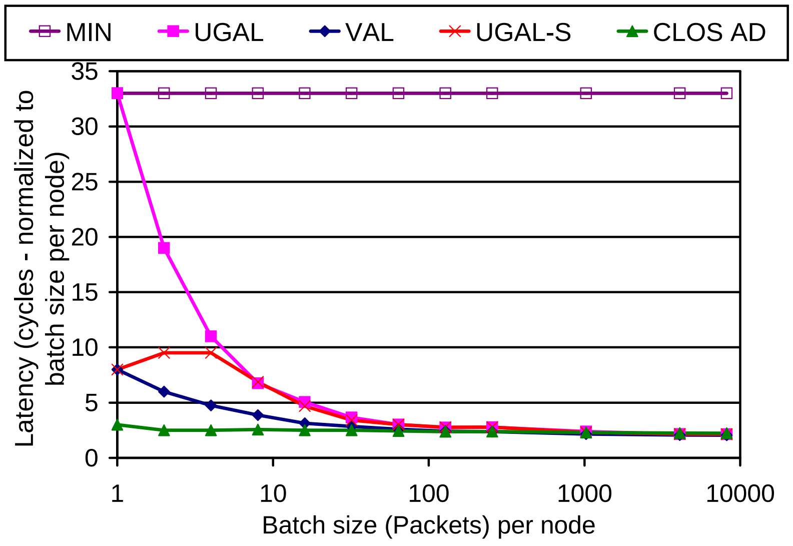


Figure 4.16: Dynamic response comparison of the routing algorithms.

small batch sizes, however, batch latency is affected by transient load imbalance. On these small batches, UGAL performs very poorly because of the greedy nature of its allocator. When the minimal queue is short, all inputs pick the minimal queue (overloading this output) before the queuing state is updated. With UGAL-S, the transient load imbalance due to greedy allocation is eliminated, but transient load imbalance across intermediate nodes remains. VAL also shares this transient load imbalance. CLOS AD eliminates both sources of transient load imbalance.

Both CLOS AD and UGAL-S require sequential allocations which can increase the router clock cycle or the pipeline depth if they are implemented sequentially. However, these algorithms can be implemented using parallel prefix schemes [49] to speed up the computation. Although CLOS AD provides performance benefit over UGAL-S, it leads to a higher complexity implementation since it requires comparing the depth of all of the non-minimal queues. Techniques to reduce the complexity of adaptive routing in high-radix Clos networks have been discussed in Section 4.1 and can be implemented on the flattened butterfly as well.

4.2.3 Comparison to Other Topologies

We compare the performance of the flattened butterfly to three other topologies : the conventional butterfly, the folded Clos, and the hypercube. We use a network of 1024 nodes – thus, the conventional butterfly is built using 2 stages of radix-32 routers, the folded-Clos network is a 3-stage network using radix-64 routers, and the hypercube is a 10 dimensional hypercube. The bisection bandwidth is held constant across the four topologies. The routing algorithms used for each of the topologies are described in Table 4.2 and the performance comparison is shown in Figure 4.17. For the different topologies, the total buffer storage is held constant i.e. the product of the VCs and the depth of each buffer is kept constant.

Topology	**Routing**	num of VCs
Flattened Butterfly	CLOS AD	2
Conventional Butterfly	destination-based	1
Folded Clos	adaptive *sequential*	1
Hypercube	e-cube	1

Table 4.2: Topology and Routing used in performance comparison. The VCs are used to break routing deadlocks.

On UR traffic (Figure 4.17(a)), all of the topologies provide 100% throughput except for the folded-Clos. By holding bisection bandwidth constant across the topologies, the folded Clos uses 1/2 of the bandwidth for load-balancing to the middle stages – thus, only achieves 50% throughput. In addition, the folded Clos has slightly higher latency because of the extra middle stage and the hypercube also has much higher latency because of its higher diameter. On WC traffic (Figure 4.17(b)), the conventional butterfly throughput is severely limited and performs identically to a flattened butterfly with minimal routing – leading to an order of magnitude difference in throughput. However, the other topologies perform similarly as they result in a throughput of 50%. Thus, the flattened butterfly provides 2x increase in performance over the folded-Clos on benign traffic while providing the same performance on the worst-case traffic pattern when the cost (i.e. bisection bandwidth) is held constant. In the next section, we provide a detailed cost model that includes the cost of the channels in different packaging hierarchy as well as the router cost and show how cost-efficient flattened butterfly is compared to the other topologies.

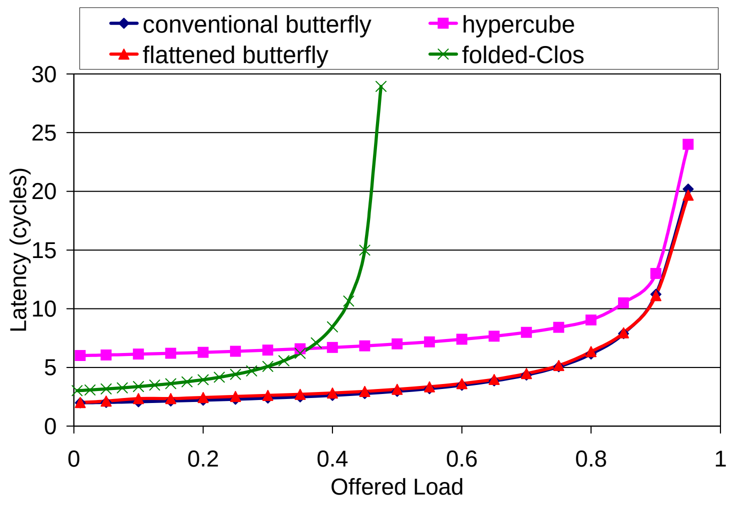
4.3 Related Work

The comparison of the flattened butterfly to the generalized hypercube was discussed earlier in Section 3.6. Non-minimal routing on the generalized hypercube has been proposed in circuit switched networks [83]. The adaptive routing proposed uses the non-minimal paths to increase path diversity but does not describe how load-balancing can be achieved with the non-minimal routes.

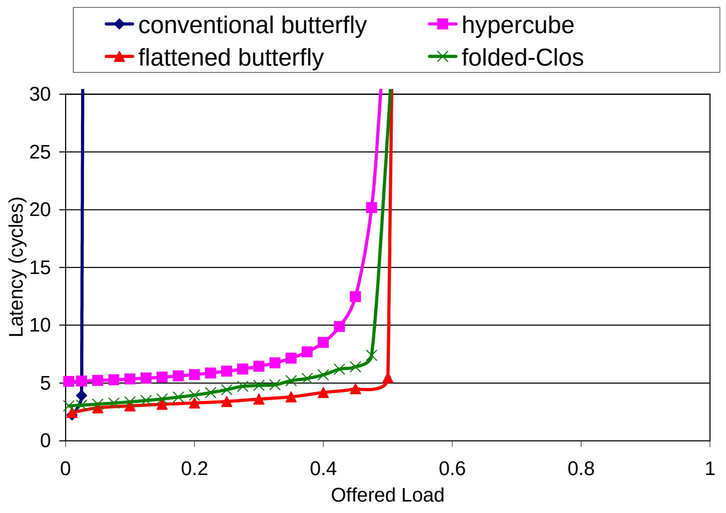
Our non-minimal routing algorithms for the flattened butterfly are motivated by recent work on non-minimal and globally adaptive routing. Non-minimal routing was first applied to torus networks and have been shown to be critical to properly load-balancing tori [73, 74]. We show that these principles and algorithms can be applied to the flattened butterfly as well. We also extend this previous work on load-balanced routing by identifying the problem of transient load imbalance in high-radix adaptive routers and show how this problem can be solved by using a sequential allocator in place of a greedy allocator and the CLOS AD routing algorithm.

4.4 Summary

In this chapter, we evaluated benefits of adaptive routing on high-radix networks. We first quantified the benefits of adaptive routing on a high-radix folded-Clos topology where all paths are minimal paths. With adaptive routing, if done properly, lower latency and less variance in the packet latency can be achieved compared to oblivious routing. Adaptive routing also achieves better buffer utilization with limited buffering and results in higher throughput. In the presence of nonuniformity in the traffic,



(a)



(b)

Figure 4.17: Topology comparisons of the flattened butterfly and the folded Clos with (a) uniform random traffic and (b) worst case traffic pattern.

adaptive routing provides significant advantages as it will "smooth out" the traffic and provide higher throughput.

Different allocation algorithms for adaptive routing in high-radix networks were also presented in this chapter. The *sequential* algorithm provides the best performance but is expensive to implement. By using randomization with the $greedy\_r(2)$ algorithm, the implementation can be simplified with minimal performance loss. The implementation complexity can be further reduced by using imprecise queue information and the latency of adaptive routing can be hidden by precomputing the adaptive routing results in a previous cycle.

The use of adaptive routing is more critical for a topology such as the flattened butterfly which includes non-minimal paths. We evaluated five routing algorithms for the flattened butterfly including both minimal and non-minimal and both adaptive and oblivious. We also compared routing algorithms using both greedy and sequential allocators. The results show that non-minimal globally-adaptive routing is necessary to load-balance the topology on both benign and adversarial traffic to provide high performance. To overcome the *transient* load imbalance that can occur in high-radix routers with greedy allocators, CLOS AD routing algorithm with a sequential allocator is required.

So far, we have ignored the router complexity caused by increasing the radix of the router. In the next chapter, we look at the router microarchitecture issues in scaling to high radix and propose an hierarchical router organization that can scale efficiently.

Chapter 5

Microarchitecture of a High-Radix Router

High-radix router design is qualitatively different from the design of low-radix high bandwidth routers. Increasing the radix of a router raises several challenges as internal switches and allocators scale as the square of the radix. This chapter addresses these challenges by proposing and evaluating alternative microarchitectures for high-radix routers. We examine the most commonly used organization of a router, the input-queued crossbar, and the different microarchitectural issues that arise when we try to scale them to high-radix routers such as switch and virtual channel allocation.

The next four sections in this chapter incrementally explore the micro-architectural space for a high-radix virtual-channel (VC) router. We start in Section 5.1 with a baseline router design, similar to that used for a low-radix router [67, 56]. We see that this design scales poorly to high radix due to the complexity of the allocators and the wiring needed to connect them to the input and output ports. In Section 5.2, we overcome these complexity issues by using distributed allocators and by simplifying virtual channel allocation. This results in a feasible router architecture, but poor performance due to head-of-line blocking. In Section 5.3, we show how to overcome the performance issues with this architecture by adding buffering at the switch crosspoints. This buffering eliminates head-of-line blocking by decoupling the input and output allocation of the switch. However, with even a modest number of virtual

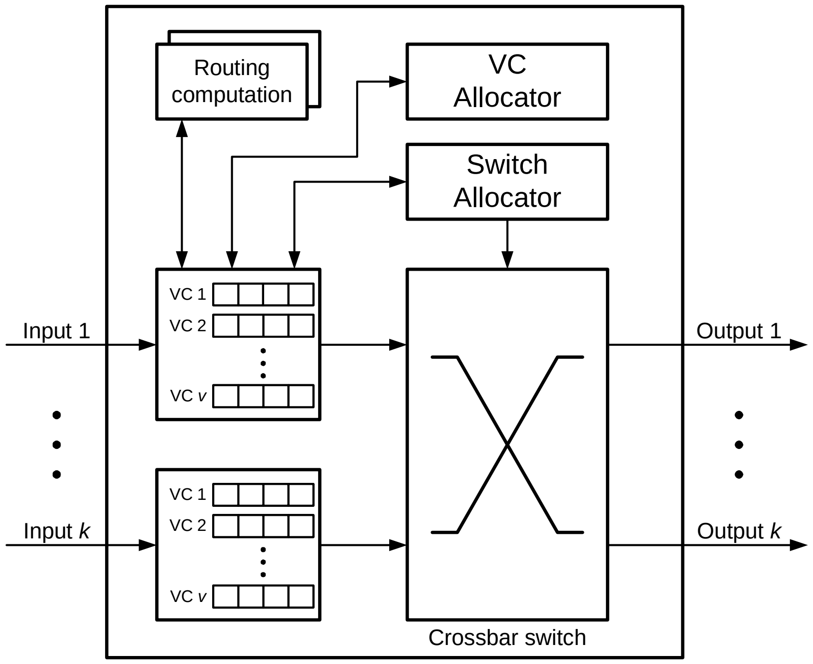


Figure 5.1: Baseline virtual channel router.

channels, the chip area required by these buffers is prohibitive. We overcome this area problem, while retaining good performance, by introducing a hierarchical switch organization in Section 5.4. Additional simulation results are presented in Section 5.5 and we conclude this chapter with a case study of the Cray YARC router [68] – a radix-64 router that implements the hierarchical switch organization.

5.1 Baseline Architecture

A block diagram of the baseline router architecture is shown in Figure 5.1. Arriving data is stored in the input buffers. These input buffers are typically separated into several parallel virtual channels that can be used to prevent deadlock, implement priority classes, and increase throughput by allowing blocked packets to be passed. The input buffers and other router resources are allocated in fixed-size units called *flits* and each packet is broken into one or more flits as shown in Figure 5.2(a).

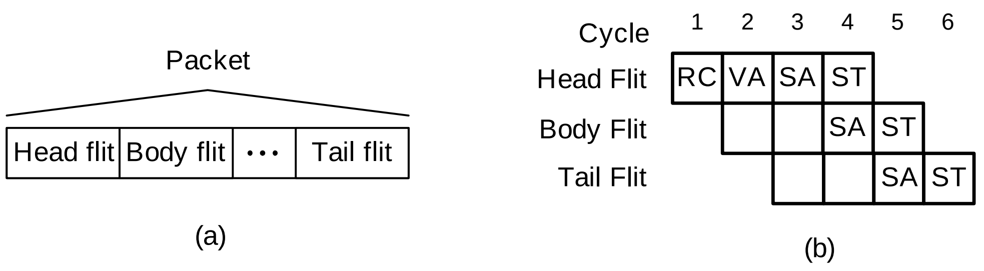


Figure 5.2: (a) Packets are broken into one or more flits (b) Example pipeline of flits through the baseline router.

The progression of a packet through this router can be separated into per-packet and per-flit steps. The per-packet actions (Steps 1 and 2) are initiated as soon as the *header flit*, the first flit of a packet, arrives:

1. Route computation (RC) - based on information stored in the header, the output port of the packet is selected.

2. Virtual-channel allocation (VA) - a packet must gain exclusive access to a downstream virtual channel associated with the output port from route computation.

   Once these per-packet steps are completed, per-flit scheduling of the packet can begin.

3. Switch allocation (SA) - if there is a free buffer in its output virtual channel, a flit can vie for access to the crossbar.

4. Switch traversal (ST) - once a flit gains access to the crossbar, it can be transferred from its input buffers to its output and on to the downstream router.

These steps (Step 3 and 4) are repeated for each flit of the packet and upon the transmission of the *tail flit*, the final flit of a packet, the virtual channel is freed and is available for another packet. A simple pipeline diagram of this process is shown in Figure 5.2(b) for a three-flit packet assuming each step takes a single cycle.

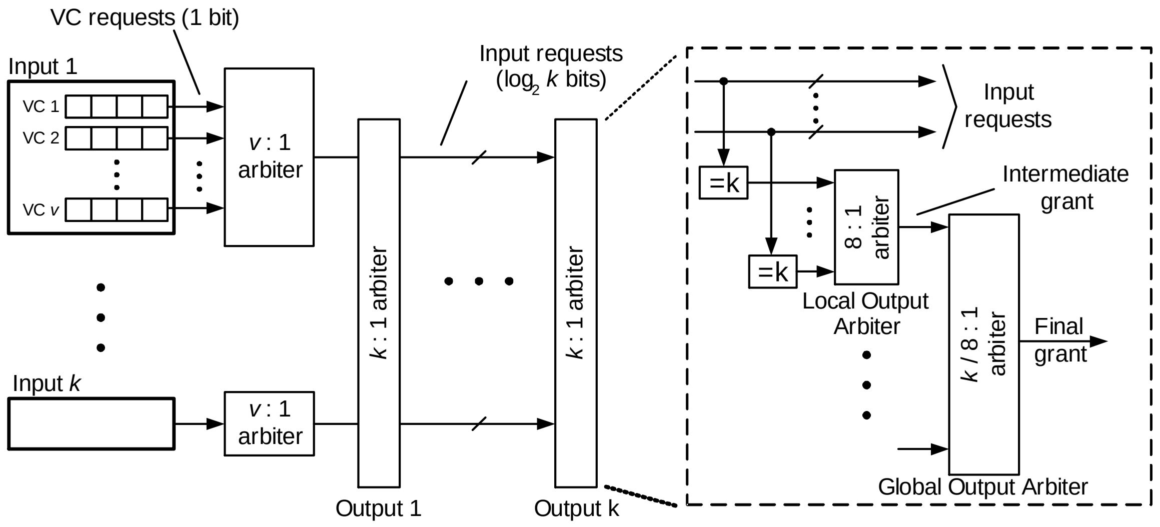


Figure 5.3: Scalable switch allocator architecture. The input arbiters are localized but the output arbiters are distributed across the router to limit wiring complexity. A detailed view of the output arbiter corresponding to output k is shown to the right.

5.2 Extending the baseline to high radix

As radix is increased, a centralized approach to allocation rapidly becomes infeasible — the wiring required, the die area, and the latency all increase to prohibitive levels. In this section, we introduce distributed structures for both switch and virtual channel allocation that scale well to high radices. In achieving this scalability, these structures compromise on performance.

5.2.1 Switch Allocation

We address the scalability of the switch allocator by using a distributed separable allocator design as shown in Figure 5.3. The allocation takes place in three stages: input arbitration, local output arbitration, and global output arbitration. During the first stage all ready virtual channels in each input controller request access to the crossbar switch. The winning virtual channel in each input controller then forwards its request to the appropriate local output arbiter by driving the binary code for the requested output onto a per-input set of horizontal request lines.

At each output arbiter, the input requests are decoded and, during stage two, each local output arbiter selects a request (if any) for its switch output from among a local group of m (in Figure 5.3, $m = 8$) input requests and forwards this request to the global output arbiter. Finally, the global output arbiter selects a request (if any) from among the k/m local output arbiters to be granted access to its switch output. For very high-radix routers, the two-stage output arbiter can be extended to a larger number of stages.

At each stage of the distributed arbiter, the arbitration decision is made over a relatively small number of inputs (typically 16 or less) such that each stage can fit in a clock cycle. For the first two stages, the arbitration is also local - selecting among requests that are physically co-located. For the final stage, the distributed request signals are collected via global wiring to allow the actual arbitration to be performed locally. Once the winning requester for an output is known, a grant signal is propagated back through to the requesting input virtual channel. To ensure fairness, the arbiter at each stage maintains a priority pointer which rotates in a round-robin manner based on the requests.

5.2.2 Virtual Channel Allocation

Virtual channel allocation (VA) poses an even more difficult problem than switch allocation because the number of resources to be allocated is multiplied by the number of virtual channels v. In contrast to switch allocation, where the availability of free downstream buffers is tracked with a credit count, with virtual channel allocation, the availability of downstream VCs is unknown. An ideal VC allocator would allow all input VCs to monitor the status of all output VCs they are waiting on. Such an allocator would be prohibitively expensive, with v^2k^2 wiring complexity.

Building off the ideas developed for switch allocation, we introduce two scalable virtual channel allocator architectures. Crosspoint virtual channel allocation (CVA) maintains the state of the output virtual channels at each crosspoint and performs allocation at the crosspoints. In contrast, output virtual channel allocation (OVA)

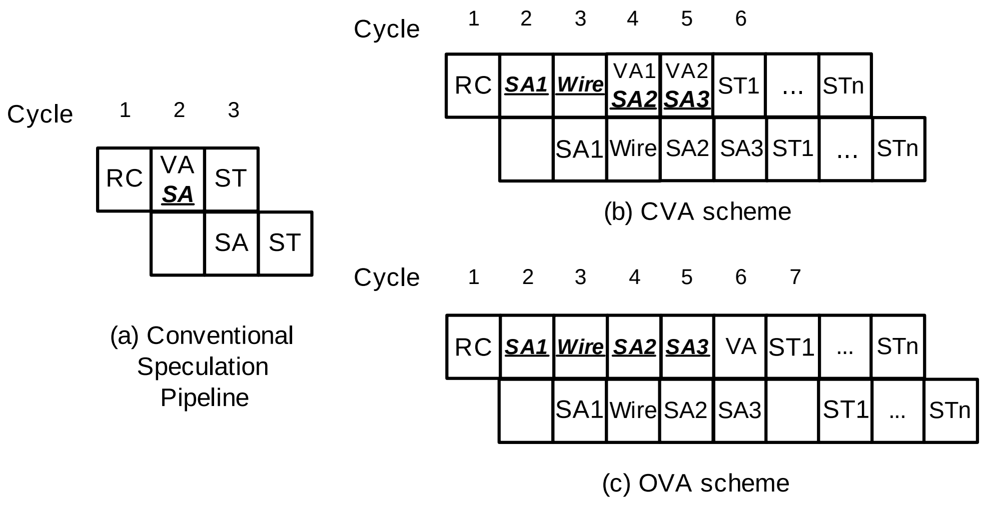


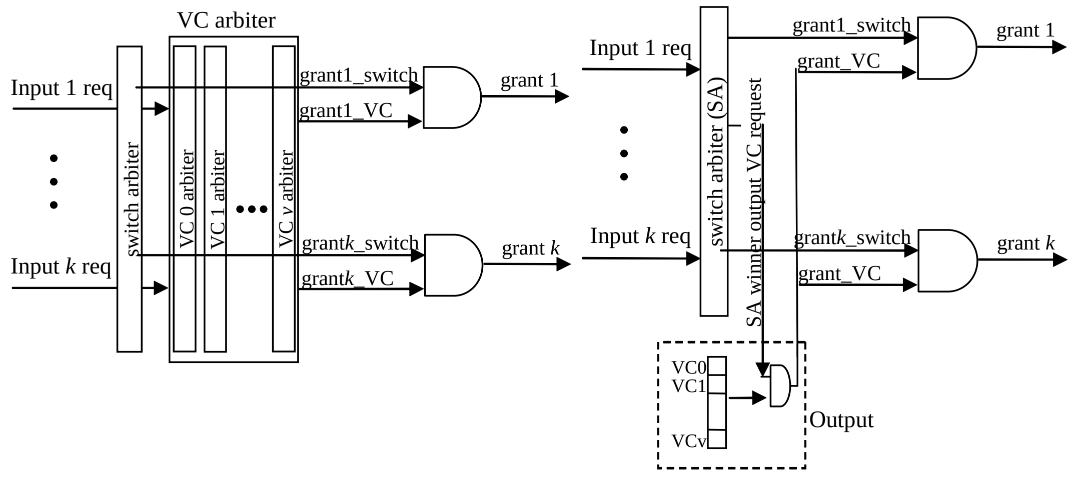
Figure 5.4: Speculative pipeline with each packet assumed to be 2 flits. (a) speculation used on the pipeline shown in Figure 5.2(b) (b) high-radix routers with CVA (c) high-radix routers with OVA. The pipeline stages underlined show the stages that are speculative.

defers allocation to the output of the switch. Both CVA and OVA involve *speculation* where switch allocation proceeds before virtual channel allocation is complete to reduce latency. Simple virtual channel speculation was proposed in [63] where the switch allocation and the VC allocation occurs in parallel to reduce the critical path through the router (Figure 5.4(a)). With a deeper pipeline in a high-radix router, VC allocation is resolved later in the pipeline. This leads to more aggressive speculation (Figure 5.4(b-c)).[1]

With CVA, VC allocation is performed at the crosspoints where the status of the output VCs is maintained. Input switch arbitration is done speculatively. Each cycle, each input controller drives a single request over a per-input set of horizontal virtual-channel-request lines to the local/global virtual output channel arbiter. Each such request includes both the requested output port and output virtual channel

The virtual channel allocator at each crosspoint includes a separate arbiter for each

[1]Pipeline key: SAx: different stages of switch allocation, Wire: separate pipeline stage for the request from the input arbiters to travel to the output arbiters, STx: switch traversal, multiple cycles will be needed to traverse the switch



(a) (b)

Figure 5.5: Block diagram of the different VC allocation schemes (a) CVA (b) OVA. In each cycle, CVA can handle multiple VC requests for the same output where as in OVA, only a single VC request for each output can be made. CVA parallelize the switch and VC allocation while in OVA, the two allocation steps are serialized. For simplicity, the logic is shown for only a single output.

output virtual channel. Instead of the k output arbiters used in the switch allocator (Figure 5.3), CVA uses a total of kv output virtual channel arbiters. Requests (if any) to each output virtual channel arbiter are decoded from the virtual channel request lines and each arbiter proceeds in the same local-global arbitration used in switch allocation.

Using OVA reduces arbiter area at some expense in performance. In this scheme, the switch allocation proceeds through all three stages of arbitration and only when complete is the status of the output virtual channel checked. If the output VC is indeed free, it is allocated to the packet. As shown in Figure 5.4(c), OVA speculates deeper in the pipeline than CVA and reduces complexity by eliminating the per-VC arbiters at each crosspoint. However, OVA compromises performance by allowing only one VC per output to be requested per allocation. A block diagram of the different VA architectures is shown in Figure 5.5 and illustrates the control logic needed for the two schemes. They are compared and evaluated in the next section.

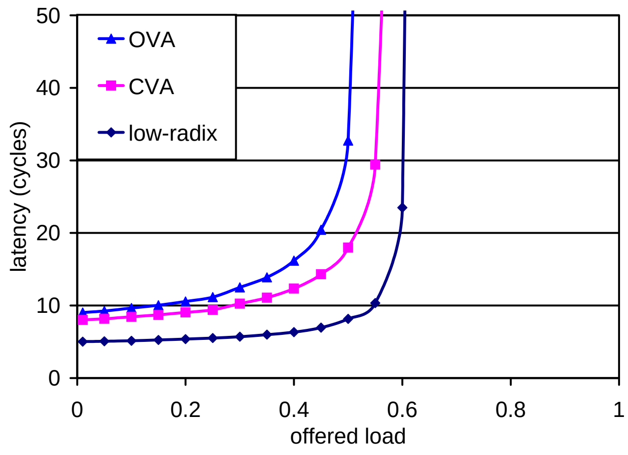


Figure 5.6: Latency vs. offered load for the baseline architecture

5.2.3 Performance

We use cycle accurate simulations to evaluate the performance of the scalable switch and virtual channel allocators. We simulate a radix-64 router using virtual-channel flow control with four virtual channels on uniform random traffic with each flit taking 4 cycles to traverse the switch. Other traffic patterns are discussed in Section 5.5. Simulation setup described earlier in Section 4.1.1 is used to evaluate alternative router microarchitectures. We begin the evaluation using single-flit packets; later, we also consider longer packets (10 flits).

A plot of latency versus offered load (as a fraction of the capacity of the switch) is shown in Figure 5.6. The performance of a low-radix router (radix 16), which follows the pipeline shown in Figure 5.2(b), with a centralized switch and virtual channel allocation is shown for comparison. Note that this represents an unrealistic design point since the centralized single-cycle allocation does not scale. Even with multiple virtual channels, head-of-line(HoL) blocking limits the low-radix router to approximately 60% throughput [39].

Increased serialization latency gives the high-radix router a higher zero-load latency than the low-radix router when considering only a single stage, as in this case. The saturation throughput of the high-radix router is approximately 50% or 12% lower than the low-radix router. The results reflect the performance of the router with realistic pipeline delays, distributed switch allocation, and a speculative virtual channel allocation. Most of this loss is attributed to the speculative VC allocation. The effect is increased when OVA is used giving a saturation throughput of about 45%.

5.2.4 Prioritized Virtual Channel Allocation

With speculative VC allocation, if the initial VC allocation fails, bandwidth can be unnecessarily wasted if the re-bidding is not done carefully. For example, consider an input queue with 4 VCs and input arbitration performed in a round-robin fashion. Assume that all of the VCs in the input queues are occupied and the flit at the head of one of the VC queues has failed VC allocation. If all 4 VCs continuously bid for the output one after the other, the speculative bids by the failed VC will waste approximately 25% of the bandwidth until the output VC it is waiting on becomes available.

Bandwidth loss due to speculative VC allocation can be reduced by giving priority in switch allocation to nonspeculative requests [63, 22]. This can be accomplished, for example by replacing the single switch allocator of Figure 5.7(a) with separate switch allocators for speculative and nonspeculative requests as shown in Figure 5.7(b). With this arrangement, a speculative request is granted bandwidth only if there are no nonspeculative requests. Prioritizing nonspeculative requests in this manner reduces bandwidth loss but at the expense of doubling switch allocation logic.

In this section we evaluate the performance gained by using two allocators to prioritize nonspeculative requests. The switch simulated in Section 5.2.3 used only a single switch allocator and did not prioritize nonspeculative requests. To ensure fairness with two switch arbiters, the priority pointer in the speculative switch arbiter is only updated after the speculative request is granted (i.e. when there are no nonspeculative requests). Our evaluation uses only 10-flit packets — with single flit packets, all flits are speculative, and hence there is no advantage to prioritizing nonspeculative flits. We prioritize nonspeculative requests only at the output switch arbiter. Prioritizing at the input arbiter reduces performance by preventing speculative flits

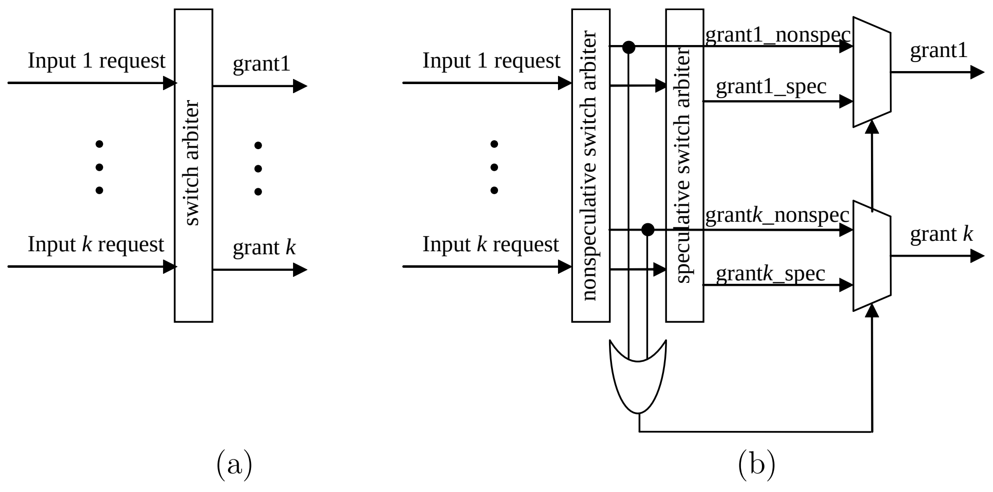
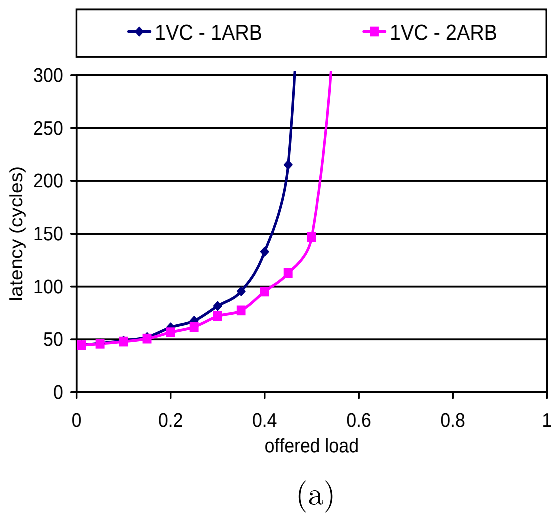


Figure 5.7: Block diagram of a switch arbiter using (a) one arbiter and (b) two arbiters to prioritize the nonspeculative requests.

representing VC requests from reaching the output virtual channel allocators.

Figure 5.8 shows that prioritizing nonspeculative requests is advantageous when there is only a single virtual channel, but has little return with four virtual channels. These simulations use CVA for VC allocation. With only a single VC, prioritized allocation increases saturation throughput by 10% and gives lower latency as shown in Figure 5.8(a). With four VCs, however, the advantage of prioritized allocation diminishes as shown in Figure 5.8(b). Here the multiple VCs are able to prevent the loss of bandwidth since with multiple VCs, a speculative request will likely find an available output VCs. Results for the OVA VC allocation follow the same trend but are not shown for space constraints. Using multiple VCs gives adequate throughput without the complexity of a prioritized switch allocator.

In the following two sections, which introduce two new architectures, we will assume the CVA scheme for VC allocation using a switch allocator without prioritization.



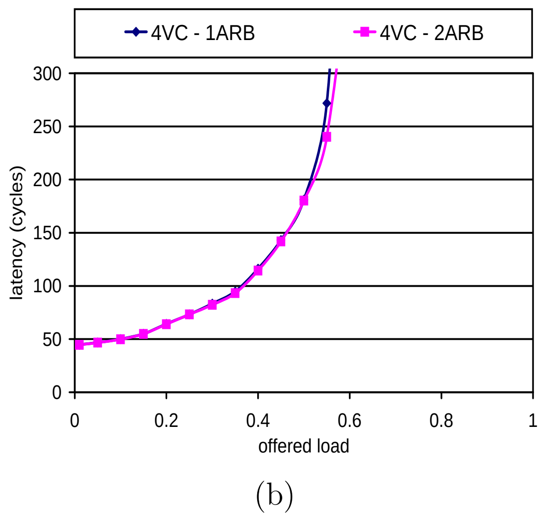


Figure 5.8: Comparison of using one arbiter and two arbiters for (a) 1VC (b) 4VC

5.3 Fully Buffered Crossbar

Adding buffering at the crosspoints of the switch (Figure 5.9(b)) decouples input and output virtual channel and switch allocation. This decoupling simplifies the allocation, reduces the need for speculation, and overcomes the performance problems of the baseline architecture with distributed, speculative allocators.

5.3.1 Switch and Virtual Channel Allocation

Input and output switch allocation are completely decoupled. A flit whose request wins the input arbitration is immediately forwarded to the crosspoint buffer corresponding to its output. At the crosspoint, local and global output arbitration are performed as in the unbuffered switch. However, because the flit is buffered at the crosspoint, it does not have to re-arbitrate at the input if it loses arbitration at the output.

The intermediate buffers are associated with the input VCs. In effect, the crosspoint buffers are per-output extensions of the input buffers. Thus, no VC allocation has to be performed to reach the crosspoint — the flit already holds the input VC.

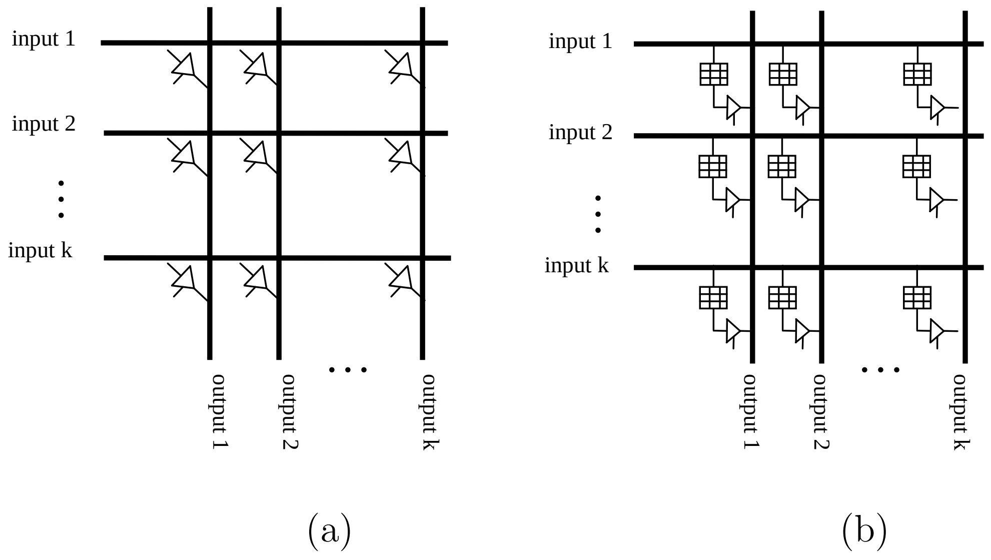


Figure 5.9: Block diagram of a (a) baseline crossbar switch and (b) fully buffered crossbar switch.

Output VC allocation is performed in two stages: a v-to-1 arbiter that selects a VC at each crosspoint followed by a k-to-1 arbiter that selects a crosspoint to communicate with the output.

5.3.2 Crosspoint buffer credits

To ensure that the crosspoint buffers never overflow, credit-based flow control is used. Each input keeps a separate free buffer counter for each of the kv crosspoint buffers in its row. For each flit sent to one of these buffers, the corresponding free count is decremented. When a count is zero, no flit can be sent to the corresponding buffer. Likewise, when a flit departs a crosspoint buffer, a credit is returned to increment the input's free buffer count. The required size of the crosspoint buffers is determined by the credit latency – the latency between when the buffer count is decremented at the input and when the credit is returned in an unloaded switch.

It is possible for multiple crosspoints on the same input row to issue flits on the same cycle (to different outputs) and thus produce multiple credits in a single cycle.

Communicating these credits back to the input efficiently presents a challenge. Dedicated credit wires from each crosspoint to the input would be prohibitively expensive. To avoid this cost, all crosspoints on a single input row share a single credit return bus. To return a credit, a crosspoint must arbitrate for access to this bus. The credit return bus arbiter is distributed, using the same local-global arbitration approach as the output switch arbiter.

We have simulated the use of a shared credit return bus and compared it with an *ideal* (but not realizable) switch in which credits are returned immediately. Simulations show that there is minimal difference between the ideal scheme and the shared bus. The impact of credit return delay is minimized since each flit takes four cycles to traverse the input row. Thus even if a crosspoint loses the credit return bus arbitration, it has 3 additional cycles to re-arbitrate for the bus without affecting the throughput.

5.3.3 Performance and area

We simulated the buffered crossbar using the same simulation setup as described in Section 5.2.3. In the switch evaluated, each crosspoint buffer contains four flit entries per virtual channel. As shown in Figure 5.10, the addition of the crosspoint buffers enables a much higher saturation throughput than the unbuffered crossbar while maintaining low latency at low offered loads. This is due both to avoiding head-of-line blocking and decoupling input and output arbitration.

With sufficient crosspoint buffers, this design achieves a saturation throughput of 100% of capacity because the head-of-line blocking is completely removed. As we increase the amount of buffering at the crosspoints, the fully buffered architecture begins to resemble an virtual-output queued (VOQ) switch where each input maintains a separate buffer for each output. The advantage of the fully buffered crossbar compared to a VOQ switch is that there is no need for a complex allocator - the simple distributed allocation scheme discussed in Section 5.2 is able to achieve 100% throughput.

To evaluate the impact of the crosspoint buffer size on performance, we vary

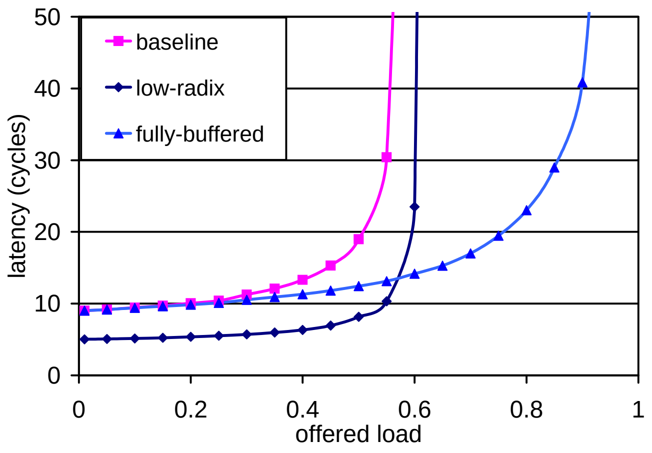


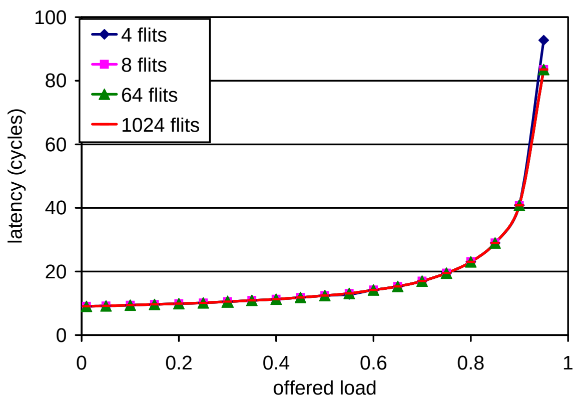
Figure 5.10: Latency vs. offered load for the fully buffered architecture. In both the fully buffered crossbar and the baseline architecture, the CVA scheme is used.

the buffer size and evaluate the performance for short and long packets. As shown in Figure 5.11(a), for short packets four-flit buffers are sufficient to achieve good performance. With long packets, however, larger crosspoint buffers are required to permit enough packets to be stored in the crosspoint to avoid head-of-line blocking in the input buffers.

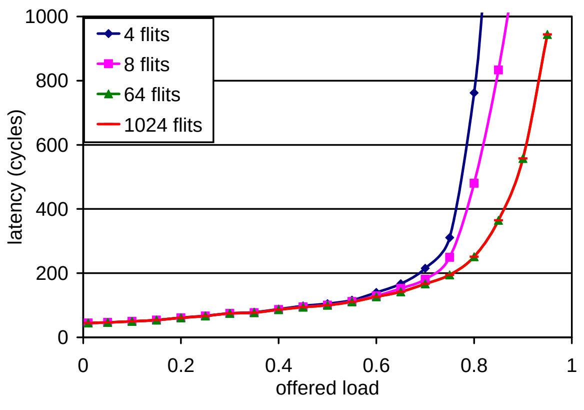
The performance benefits of a fully-buffered switch come at the cost of a much larger router area. The crosspoint buffering is proportional to vk^2 and dominates chip area as the radix increases. Figure 5.12 shows how storage and wire area grow with k in a $0.10\mu m$ technology for v=4. The storage area includes crosspoint and input buffers. The wire area includes area for the crossbar itself as well as all control signals for arbitration and credit return. As radix is increased, the bandwidth of the crossbar (and hence its area) is held constant. The increase in wire area with radix is due to increased control complexity. For a radix greater than 50, storage area exceeds wire area.

5.3.4 Fully Buffered Crossbar without per-VC buffering

One approach to reducing the area of the fully buffered crossbar is to eliminate per-VC buffering at the crosspoints. With a single shared buffer among the VCs per



(a)



(b)

Figure 5.11: Latency vs. offered load for the fully buffered architecture for (a) short packet (1 flit) and (b) long packet (10 flit) as the crosspoint buffer size is varied

crosspoint, the total amount of storage area can be reduced by a factor of v. This approach would still decouple the input and the output switch arbitration, thus providing good performance over a non-buffered crossbar. However, VC allocation is complicated by the shared buffers and it presents new problems.

As discussed in Section 5.2.2, VC allocation is performed speculatively in order to reduce latency — the flit is sent to the crosspoint without knowing if the output VC can be allocated. With per input VC crosspoint buffers, this was not an issue. However, with a crosspoint buffer shared between input VCs, a flit cannot be allowed to stay in the crosspoint buffer while awaiting output VC allocation. If the speculative flit is allowed to wait in the crosspoint buffer following an unsuccessful attempt at VC allocation, this flit can block all input VCs. This not only degrades performance but also creates dependencies between VCs that may lead to deadlock. Because flits cannot wait for VC allocation in the crosspoint buffers, speculative flits that have been sent to the crosspoint switch must be kept in the input buffer until an ACK is received from output VC allocation. If the flit fails VC allocation or if there are no downstream buffers, the flit is removed from the buffer at the crosspoint and a NACK is sent back to the input, and the input has to resend this flit at a later time. Note that with per-VC buffers, this blocking does not occur and no ACKs are necessary.

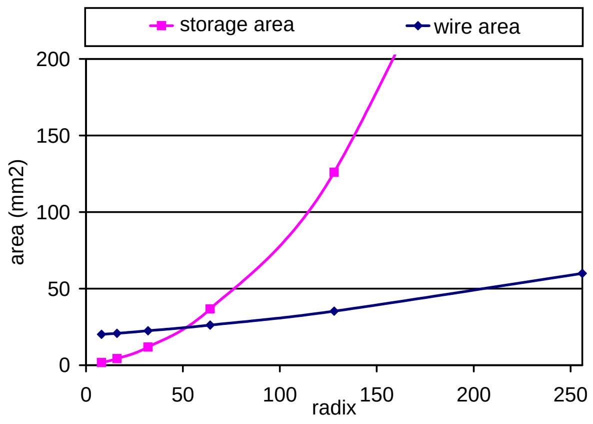


Figure 5.12: Area comparison between storage area and wire area in the fully buffered architecture.

5.3.5 Other Issues

The fully-buffered crossbar presents additional issues beyond the quadratic growth in storage area. This design requires k^2 arbiters, one at each crosspoint, with v inputs each to arbitrate between the VCs at each crosspoint. In addition, each input needs a kv entry register file to maintain the credit information for the crosspoint buffers and logic to increment/decrement the credit information appropriately.

Besides the microarchitectural issues, the fully buffered crossbar restricts the routing algorithm that can be implemented. A routing relation may return multiple outputs as a possible next hop. With a fully buffered architecture and the distributed allocators, multiple outputs can not be requested simultaneously and only one output port can be selected. The hierarchical approach that we present in the next section provides a solution that is a compromise between a centralized router and the fully buffered crossbar.

5.4 Hierarchical Crossbar Architecture

A block diagram of the hierarchical crossbar is shown in Figure 5.13. The hierarchical crossbar is built by dividing the crossbar switch into subswitches where only the inputs and outputs of the subswitch are buffered. A crossbar switch with k ports that has a

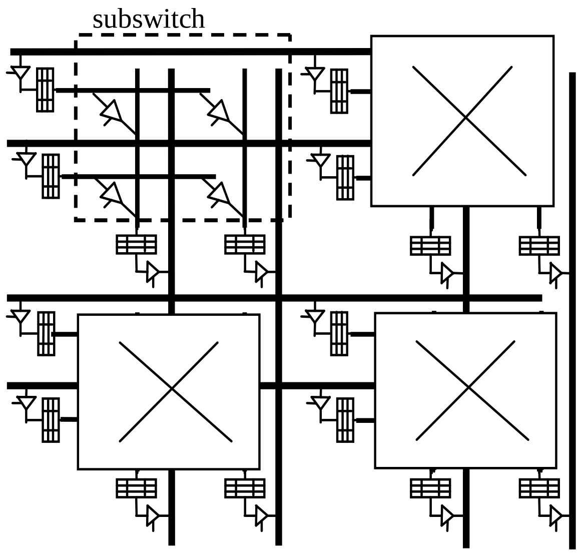


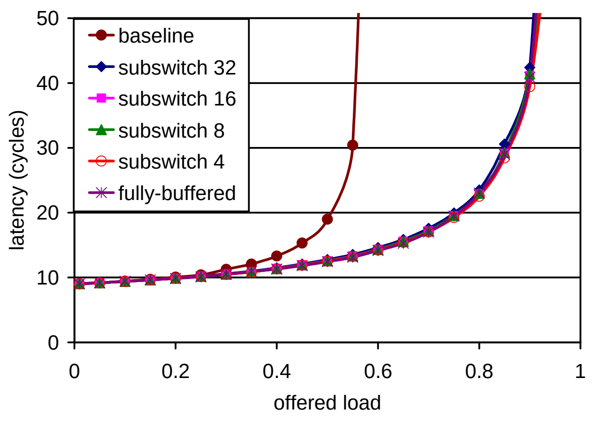
Figure 5.13: Hierarchical Crossbar (k=4) built from smaller subswitches (p=2).

subswitch of size p is made up of $(k/p)^2$ $p \times p$ crossbars, each with its own input and output buffers.

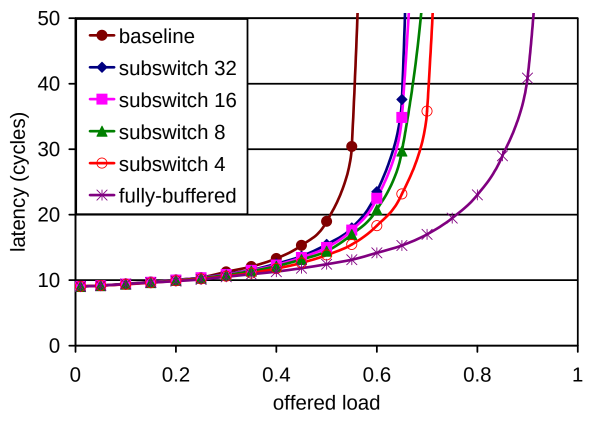
By implementing a subswitch design the total amount of buffer area grows as $O(vk^2/p)$, so by adjusting p the buffer area can be significantly reduced from the fully-buffered design. This architecture also provides a natural hierarchy in the control logic — local control logic only needs to consider information within a subswitch and global control logic coordinates the subswitches.

Similar to the fully-buffered architecture, the intermediate buffers on the subswitch boundaries are allocated on a per-VC basis. The subswitch input buffers are allocated according to a packet's *input* VC while the subswitch output buffers are allocated according to a packet's *output* VC. This decoupled allocation reduces HoL blocking when VC allocation fails and also eliminates the need to NACK flits in the intermediate buffers. By having this separation at the subswitches with buffers, it divides the VC allocation into a local VC allocation within the subswitch and a global VC allocation among the subswitches.

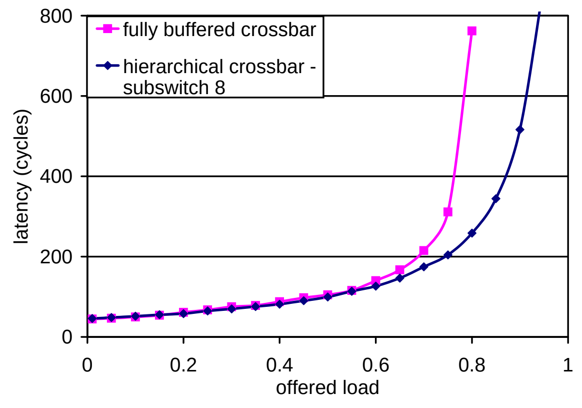
With the hierarchical design, an important design parameter is the size of the



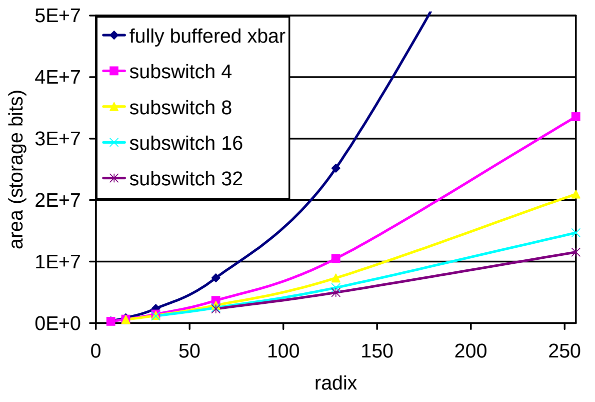
(a)



(b)



(c)



(d)

Figure 5.14: Comparison of the hierarchical crossbar as the subswitch size is varied (a) uniform random traffic (b) worst-case traffic (c) long packets and (d) area. k=64 and v=4 is used for the comparison.

subswitch, p which can range from 1 to k. With small p, the switch resembles a fully-buffered crossbar resulting in high performance but also high cost. As p approaches the radix k, the switch resembles the baseline crossbar architecture giving low cost but also lower performance.

The throughput and area of hierarchical crossbars with various subswitch sizes are compared to the fully buffered crossbar and the baseline architecture in Figure 5.14. On uniform random traffic(Figure 5.14(a)), the hierarchical crossbar performs as well as the fully buffered crossbar, even with a large subswitch size. With uniform random traffic, each subswitch sees only a fraction of the load — $\frac{\lambda}{k/p}$ where λ is the total offered load. Even with just two subswitches, the maximum load seen by any subswitch for uniform random traffic pattern will always be less than 50% and the subswitches will not be saturated.

A worst-case traffic pattern for the hierarchical crossbar concentrates traffic on a small number of subswitches. For this traffic pattern, each group of (k/p) inputs that are connected to the same row of subswitches send packets to a randomly selected output within a group of (k/p) outputs that are connected to the same column of subswitches. This concentrates all traffic into only (k/p) of the $(k/p)^2$ subswitches. Figure 5.14(b) shows performance on this traffic pattern. The benefit of having smaller subswitch size is apparent. On this worst-case pattern, the hierarchical crossbar does not achieve the throughput of the fully-buffered crossbar (about 30% less throughput for $p = 8$). However hierarchical crossbars outperforms the baseline architecture by 20% (for $p = 8$). Fortunately, this worst-case traffic pattern is very unlikely in practice.

Like the fully-buffered crossbar, the throughput of the hierarchical crossbar on long packets depends on the amount of intermediate buffering available. The evaluation so far assumed that each buffer in the hierarchical crossbar holds four flits. In order to provide a fair comparison, we keep the *total* buffer size constant and compare the performance of the hierarchical crossbar with the fully buffered crossbar on long packets. The fully buffered crossbar has 4 entries per crosspoint buffer while the hierarchical crossbar($p = 8$) has 16 entries per buffer.[2] Figure 5.14(c) compares the performance of a fully-buffered crossbar with a hierarchical crossbar ($p = 8$) with equal total buffer space. Under this constraint, the hierarchical crossbar provides better throughput on uniform random traffic than the fully-buffered crossbar.

The cost of the two architectures, in terms of area, is compared in Figure 5.14(d). The area is measured in terms of the storage bits required in the architecture. As radix increases, there is quadratic growth in the area consumed by the fully buffered crossbar. For $k = 64$ and $p = 8$, a hierarchical crossbar takes 40% less area than a

[2]To make the total buffer storage equal, each input and output buffer in the hierarchical crossbar has $p/2$ times the storage of a crosspoint buffer in the fully-buffered crossbar.

Name	**Description**
diagonal traffic	traffic pattern where input i send packets only to output i and $(i+1) \bmod k$
hotspot	uniform traffic pattern with $h = 8$ outputs being oversubscribed. For each input, 50% of the traffic is sent to the h outputs and the other 50% is randomly distributed.
bursty	uniform traffic pattern is simulated with a bursty injection based on a Markov ON/OFF process and average burst length of 8 packets is used.

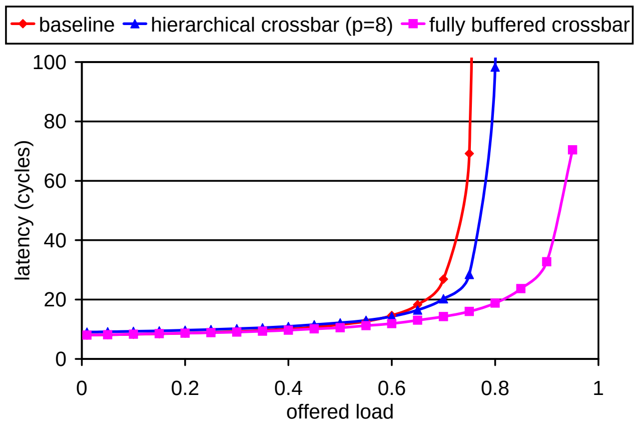
Table 5.1: Nonuniform traffic pattern evaluated.

fully-buffered crossbar.

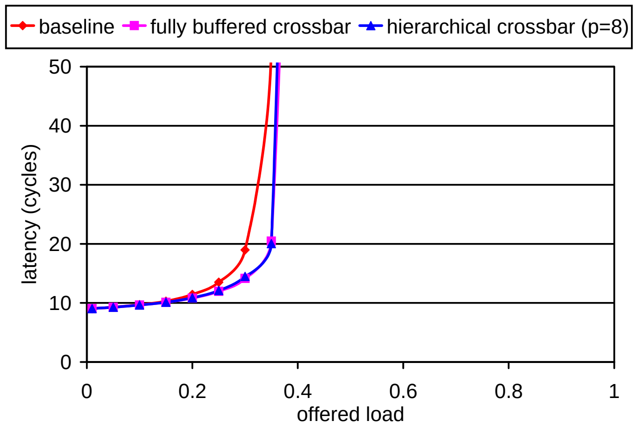
5.5 Simulation Results

In addition to uniform random traffic, we present additional simulations to compare the architectures presented using traffic patterns summarized in Table 5.1. The results of the simulations are shown in Figure 5.15. On diagonal traffic, the hierarchical crossbar exceeds the throughput of the baseline by 10%. Hotspot traffic limits the throughput to under 40% capacity for all three architectures. At this point the oversubscribed outputs are saturated. The hierarchical crossbar and the fully-buffered crossbar achieve nearly 100% throughput on bursty traffic while the baseline architecture saturates at 50%. The hierarchical crossbar outperforms the full-buffered crossbar on this pattern. It is better able to handle bursts of traffic because it has two stages of buffering, at both the inputs and the outputs of each subswitch, even though it has less total buffering than the fully-buffered crossbar.

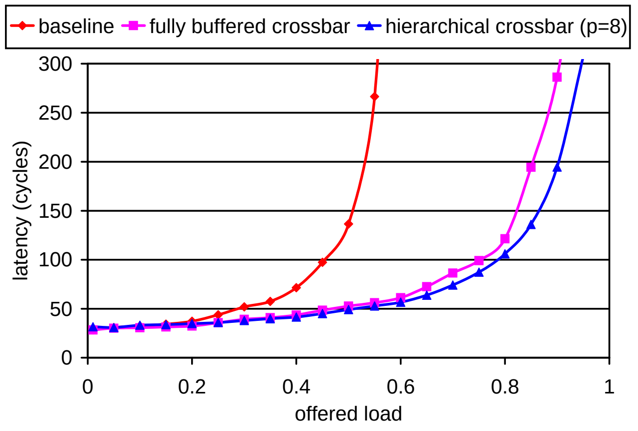
While a single high-radix router has higher zero-load latency than a low-radix router (Figure 5.6), this factor is more than offset by the reduced hop-count of a high-radix network giving lower zero-load latency for the network as a whole. Latency as a function of the offered load for a network of 4096 nodes with both radix-64 and radix-16 routers is shown in Figure 5.16. Both routers use the hierarchical architecture proposed in Section 5.4. The routers are configured as a Clos[17] network with three



(a)



(b)



(c)

Figure 5.15: Performance comparison on non-uniform traffic patterns (a) diagonal traffic (b) hotspot traffic (c) bursty traffic. Parameters used are k=64, v=4, and p=8 with 1 flit packets

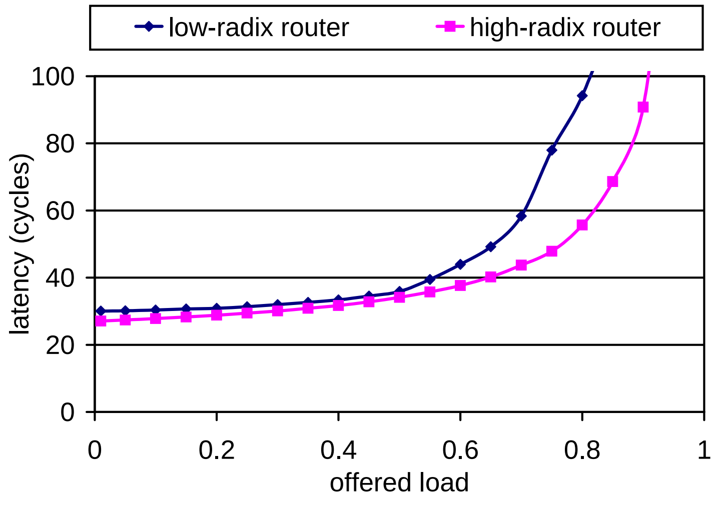


Figure 5.16: Network simulation comparison

stages for the radix-64 routers and five stages for the radix-16 routers. The simulation was run using an oblivious routing algorithm (middle stages are selected randomly) and uniform random traffic.

5.6 Case Study: Cray BlackWidow YARC router

The Cray BlackWidow vector multiprocessor system [2] is one of the first systems to implement a high-radix network. The topology in the BlackWidow network is a variant of a high-radix folded-Clos topology, and the high-radix router microarchitecture is based on the hierarchical organization described earlier in this chapter. The details of the BlackWidow network and the YARC router used in the network can be found in [68]. In this section, we highlight some of the key differences between the YARC implementation and the hierarchical crossbar organization described earlier in Section 5.4.

A block diagram of the YARC router is shown in Figure 5.17 and a die photo of YARC is shown in Figure 5.18. The YARC router is a radix-64 router and the implementation is partitioned into 64 tiles with each tile containing an 8×8 subswitch, an input and an output port, and associated buffers which consist of input buffers,

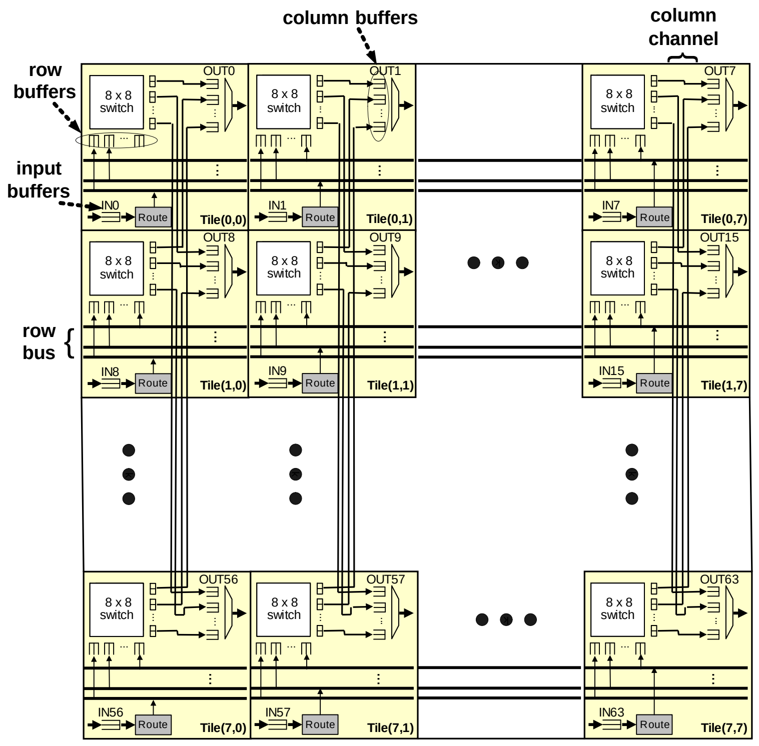


Figure 5.17: Block diagram of the Cray YARC router.

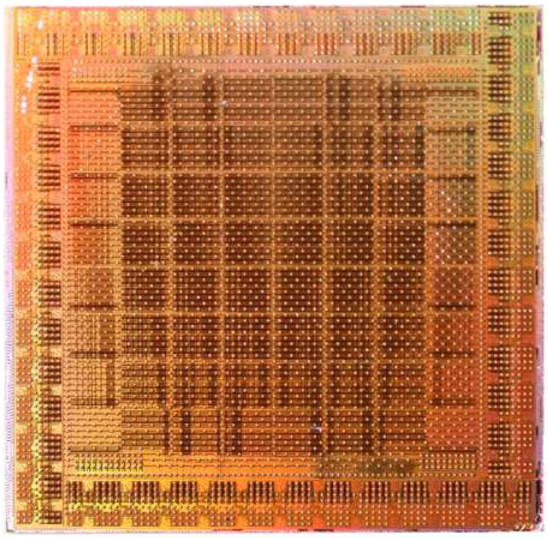


Figure 5.18: Die photo of the Cray YARC router (Courtesy Dennis Abts of Cray).

row buffers, and column buffers. The tiles communicate with other tiles through the row bus and the column channels. The tiled organization of the high-radix router led to a *complexity-effective* design as only a single design of a tile is required and is duplicated across the router.

Both the hierarchical organization (Section 5.4) and the YARC router provide an input speedup since each input port is connected to all subswitches in its row. However, the YARC router exploits the abundant wire resources available on-chip as output speedup is provided from the subswitches – i.e., the outputs of the subswitch are connected to all the outputs in each column. With the large number ports in a high-radix router, the output arbitration needs to be broken into multiple stages and the YARC router also performs output arbitration in two stages. The first stage arbitrates for the outputs of the subswitches and the second stage arbitrates for the output ports among the subswitches' outputs in each column. However, by providing output speedup, the output arbitration is simplified because the arbiter is local to the output port rather than being a central, shared resource. The YARC implementation can be viewed as a two-stage network as shown in Figure 5.19 – the first stage consisting of the input speedup to the subswitches and the second stage consisting of output speedup to the output ports. Similar to a crossbar, there is only a single path between an input and an output port but an 8$\times$ speedup is provided at both the input and the output ports.

Although there are abundant amount of wire resources available on-chip, the buffering available on-chip to implement the YARC router microarchitecture is limited. Thus, the intermediate buffers (row buffers and the column buffers) are area-constrained and the number of entries in these buffers are limited. As a result, although virtual cut-through flow control is implemented across chips in the Black-Widow network, wormhole flow control is implemented within the YARC router – across row buffers and column buffers.

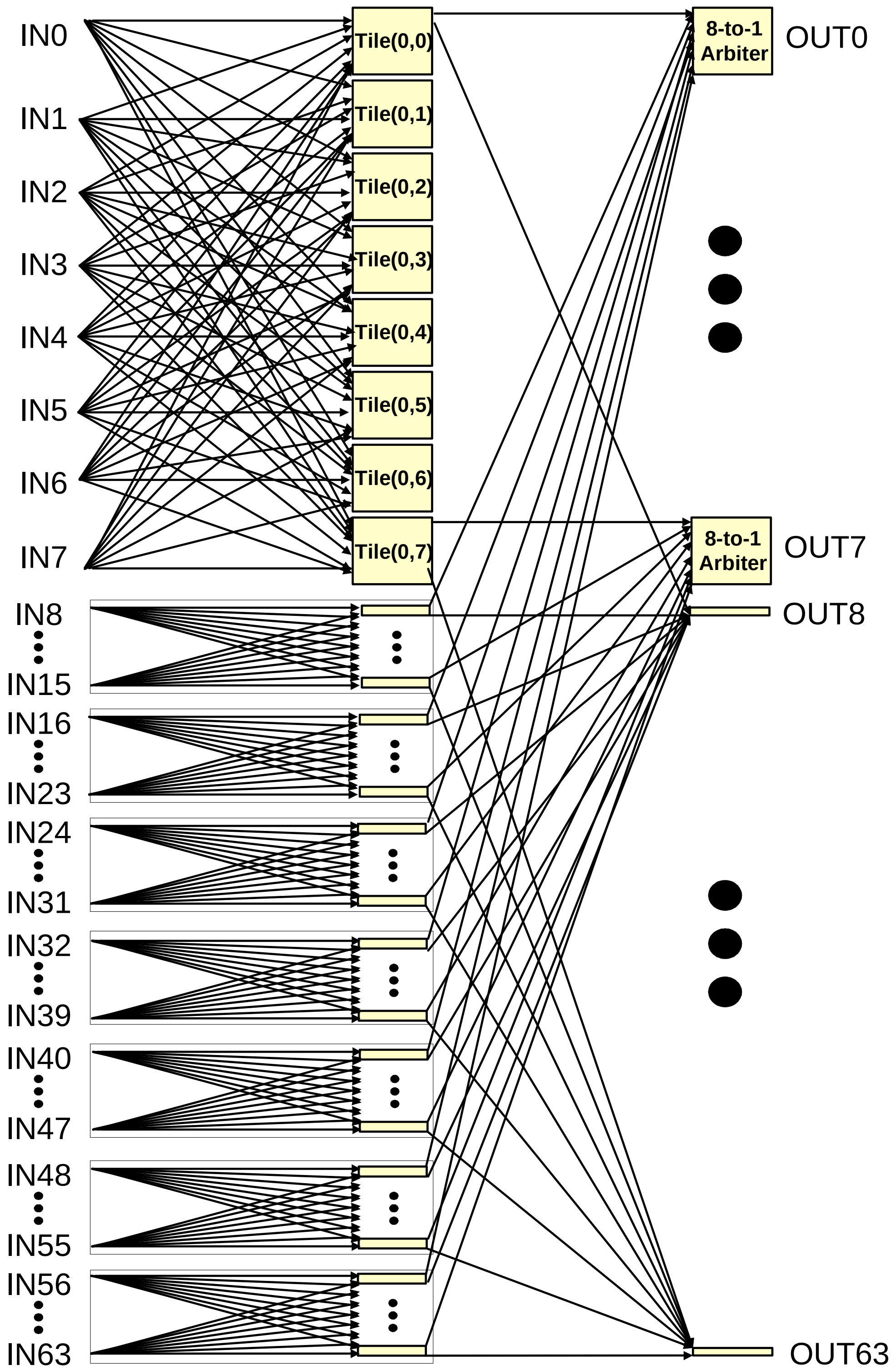


Figure 5.19: Block diagram of the Cray YARC router illustrating the internal speedup.

5.7 Related Work

Most existing single chip router architectures are designed for small radix implementations [67, 56]. Commodity routers such as Quadrics [9] implement a radix-8 router and the highest radix available from Myrinet is radix-32 [58]. The IBM SP2 switch [75] is a radix-8.

The scaling issue of switches have been addressed in IP routers in order to support the increasing line rate. The IBM Prizma architecture third generation switch [28] has increased the number of ports from 32 to 64. To overcome the limitation of the scheduling or the allocation in these IP routers, buffered crossbars [66] have been studied which decouple the input and the output allocation and has been shown to achieve high performance. However, the routers for these switch fabrics are fundamentally different. IP routers have packets that are significant longer (usually at least 40B to 100B) compared to the packet size of a shared memory multiprocessors (8B to 16B). Thus, IP routers are less sensitive to latency in the design than the high-radix routers used in a multi-computer system. In addition, these IP routers are often built using multiple chips and thus, do not have the area constraint that is present in our design.

High-radix crossbars have been previously designed using multiple lower-radix crossbars. A design implemented with multiple chips and each chip acting as a subcrossbar is outlined in [22]. Other work has attempted to exploit traffic characteristics to partition the crossbar into smaller, faster subcrossbars [16] but does not scale to high radix. A two-dimensional crossbar with VOQs has been decomposed to a switch organization similar to our hierarchical router [38]. However, these references neither discuss how to scale the allocation schemes to high-radix nor do they provide the per-VC intermediate buffering at the subswitches.

5.8 Summary

Existing microarchitectures for building routers do not scale to high radix and in this chapter, we presented alternative router microarchitecture that scale to high radix. Naive scaling of a baseline architecture provides a simple design but results in less than 50% throughput. A fully-buffered crossbar with per-VC buffering at the crosspoints achieves nearly 100% throughput but at a prohibitive cost. Thus, we proposed an alternative architecture, the hierarchical crossbar, that maintains high performance but at a lower cost. The hierarchical crossbar provides a 20-60% increase in the throughput over the baseline architecture and results in a 40% area savings compared to the fully buffered crossbar. The hierarchical nature of the architecture provides the benefit of logically separating the control logic in a hierarchical manner as well. As a case study, we also presented the microarchitecture of the radix-64 Cray YARC router that implements a variation of the proposed hierarchical router microarchitecture.

In the next chapter, we present how we can extend the benefits of high-radix routers to on-chip networks. By using high-radix routers and the flattened butterfly topology, we describe an on-chip network that is more efficient than conventional on-chip networks such as a 2-D mesh network.

Chapter 6

High-Radix Routers in On-Chip Networks

Up to this point, we have extensively studied the design of high-radix networks in *off*-chip networks. In this chapter, we examine how high-radix routers can be used in on-chip networks. Although on-chip networks exhibit very different constraints from off-chip networks, they share similar design goals, which include high performance (low latency) and low cost. Leveraging the flattened butterfly topology proposed earlier in Chapter 3, we describe how the topology can be mapped to on-chip networks to provide a more efficient network compared to a conventional 2-D mesh network. In addition, we describe how the flattened butterfly topology for on-chip networks can exploit the planar VLSI layout and be augmented to allow non-minimal routing without traversing non-minimal physical distances – further reducing latency as well as energy consumption. This can be achieved by adding bypass muxes to the network to improve the efficiency of using *bypass* channels.

6.1 Background

Chip multiprocessors are becoming more widely used to efficiently use the increasing number of transistors available in a modern VLSI technology. As the number of cores increases in such architectures, an on-chip network is used to connect the cores to provide a communication substrate. These on-chip networks should provide both low latency and high bandwidth communication fabrics that efficiently support a diverse variety of workloads.

Most on-chip networks that have been proposed are low-radix, mostly using a 2-D mesh such as the networks found in the RAW processor [76], the TRIPS processor [31], the 80-node Intel's Teraflops research chip [78], and the 64-node chip multiprocessor from Tilera [5]. Although such low-radix networks provide a very simple network and lead to very short wires in the architecture, these networks have several disadvantages which include long network diameter as well as energy inefficiency because of the extra hops. By reducing the diameter of the network, high-radix networks are advantageous both for latency and power since delay due to intermediate routers is greatly reduced as well as the power consumed by intermediate routers.

In this section, we describe how the cost structures of on-chip networks differ from system interconnection networks and presents an argument for high-radix networks on chip.

6.1.1 Cost of On-Chip Networks

The cost of an off-chip interconnection network typically increases with the channel count. As the channel count increases with the hop count, reducing the diameter of the network reduces the cost of the network [45]. On-chip networks differ because bandwidth is plentiful because of inexpensive wires, while buffers are comparatively expensive. However, reducing the diameter of on-chip networks remains beneficial for several reasons. The power consumed by the channels is a significant part of aggregate on-chip network power consumption [8]; thus, reducing the number of channels can reduce the overall power consumption. Furthermore, since buffered flow control is often used in on-chip networks, the aggregate area allocated to the input buffers in the routers decreases as the number of channels is reduced. In this work, we show how the use of high-radix routers and the flattened butterfly topology leads to lower diameter and as a result, leads to lower cost by reducing the number of channels and the amount of buffers required in the network.

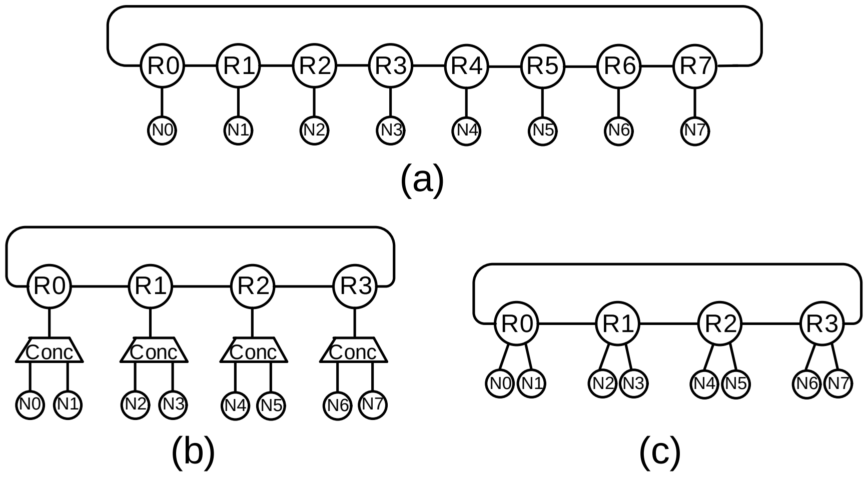


Figure 6.1: The use of concentration in interconnection networks – (a) 8 node (N0 - N7) ring with 8 routers (R0 - R7) without concentration, (b) 4 node ring with 2-way concentrator and (c) the same topology as (b) with the 2-way concentrator integrated into the router.

In addition to reducing the diameter, the use of *concentration*, where network resources are shared among different processor nodes, can improve the efficiency of the network [22]. An example of concentration is shown in Figure 6.1. Using a ring topology, 8 nodes can be connected with 8 routers as shown in Figure 6.1(a). By using a concentration factor of two, the 8 nodes can be connected in a 4 *node* ring where each *node* consists of two terminal nodes and a router, as shown in Figure 6.1(b). The use of concentration aggregates traffic from different nodes into a single network interface. This reduces both the number of resources allocated to the network routers and the average hop count, which can improve latency. Thus, while providing the same bisection bandwidth, concentration can reduce the cost of the network by reducing the network size. The concentrator can be integrated into the router by increasing the radix of the router, as shown in Figure 6.1(c), to allow all of the terminal nodes to access the network concurrently, instead of allowing only one terminal node associated with a router to access the network during any one cycle. The use of concentration in on-chip networks also reduces the wiring complexity as shown in Section 6.7.1.

Concentration is practical for on-chip networks because the probability that more than one of the processors attached to a single router will attempt to access the

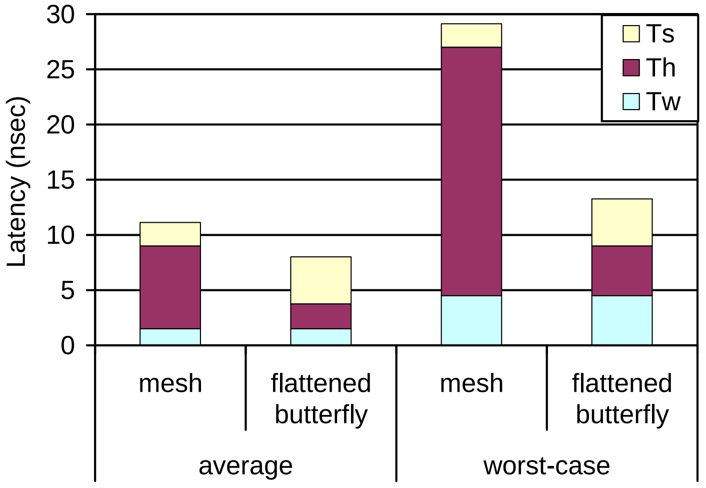


Figure 6.2: Latency of a packet in on-chip networks.

network on a given cycle is relatively low. For example, in a chip multiprocessor (CMP) architecture with a shared L2 cache distributed across the chip, the L1 miss rate is often under 10% [15], which results in relatively low traffic injection rates at the processors. Consequently, using concentration to share the network resources is an effective technique for CMP traffic. The advantages of using concentration in on-chip network were previously described for 2-D mesh networks [8]. In this paper, we describe how the flattened butterfly topology [43] for on-chip networks use concentration as well as high-radix routers to reduce the diameter of the network to improve cost-efficiency.

6.1.2 Latency in On-Chip Networks

Latency is a critical performance metric for on-chip networks. As described earlier in Section 2.2, the latency of a packet through an interconnection network can be expressed as the sum of the header latency (T_h), the serialization latency (T_s), and the time of flight on the wires (T_w),

$$\begin{aligned} T &= T_h + T_s + T_w \\ &= Ht_r + L/b + T_w \end{aligned}$$

where t_r is the router delay, H is the hop count, L is the packet size, and b is the channel bandwidth.

Minimizing latency requires establishing a careful balance between T_h and T_s. For on-chip networks, wires are abundant, on-chip bandwidth plentiful, and consequently T_s can be reduced significantly by providing very wide channels. However, traditional 2-D mesh networks tend to establish T_s and T_h values that are unbalanced, with wide channels providing a low T_s while T_h remains high due to the high hop-count. Consequently, these networks fail to minimize latency. For example, in the 2-D mesh network used in the Intel TeraFlop [78], with uniform random traffic, T_h is approximately 3 times T_s and for worst case traffic, there is approximately $10\times$ difference. [1] However, by using high-radix routers and the flattened butterfly topology, the hop count can be reduced at the expense of increasing the serialization latency, assuming the bisection bandwidth is held constant, as is shown in Figure 6.2. As a result, the flattened butterfly achieves a lower overall latency. Note that the wire delay (T_w) associated with the Manhattan distance between the source and the destination nodes generally corresponds to the minimum packet latency in an on-chip network [48]. This paper describes how high-radix routers and the flattened butterfly topology can be used the build on-chip networks that try to approach this ideal latency by significantly reducing the number of intermediate routers.

6.2 Topology Description

As described earlier in Chapter 3, the flattened butterfly topology is a cost-efficient topology that exploits high-radix routers. To map a 64-node on-chip network onto the flattened butterfly topology, we collapse a 3-stage radix-4 butterfly network (4-ary 3-fly) to produce the flattened butterfly shown in Figure 6.3(a). The resulting network has 2 dimensions and uses radix-10 routers. With four processor nodes attached to a router, the routers have a concentration factor of 4. The remaining 6 router ports are used for inter-router connections: 3 ports are used for the dimension 1 connections,

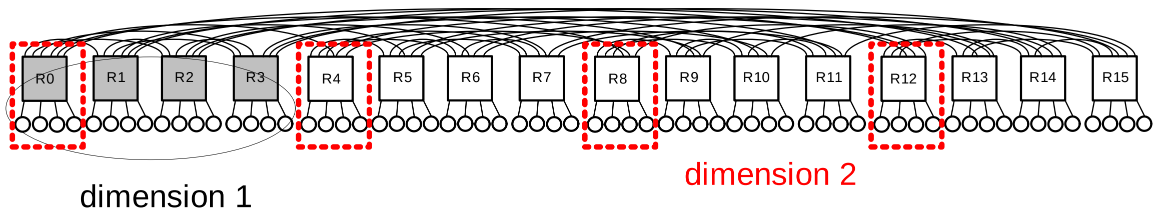
[1] The latency calculations were based on Intel TeraFlop [78] parameters ($t_r = 1.25$ns, $b = 16$GB/s, $L = 320$bits) and estimated value of wire delay for 65nm ($t_w = 250$ps per mm).

and 3 ports are used for the dimension 2 connections. Routers are placed as shown in Figure 6.3(b) to embed the topology in a planar VLSI layout. Routers connected in dimension 1 are aligned horizontally, while routers connected in dimension 2 are aligned vertically; thus, the routers within a row are fully connected, as are the routers within a column.

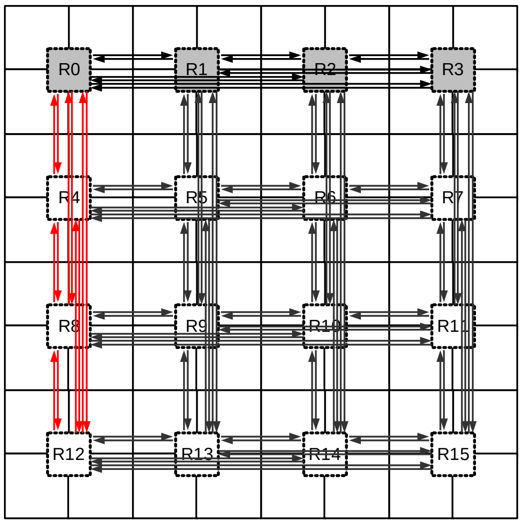
The wire delay associated with the Manhattan distance between a packet's source and its destination provides a lower bound on latency required to traverse an on-chip network. When minimal routing is used, processors in this flattened butterfly network are separated by only 2 hops, which is a significant improvement over the hop count of a 2D mesh. The flattened butterfly attempts to approach the wire delay bound by reducing the number of intermediate routers – resulting in not only lower latency but also lower energy consumption. However, the wires connecting distant routers in the flattened butterfly network are necessarily longer than those found in the mesh. The adverse impact of long wires on performance is readily reduced by optimally inserting repeaters and pipeline register to preserve the channel bandwidth while tolerating channel traversal times that may be several cycles.

6.3 Routing and Deadlock

Both minimal and non-minimal routing algorithms can be implemented on the flattened butterfly topology. A limited number of virtual channels (VCs) [23] may be needed to prevent deadlock within the network when certain routing algorithms are used. Additional VCs may be required for purposes such as separating traffic into different classes or avoiding deadlock in the client protocols. Dimension-ordered routing (DOR) can be used as a minimal routing algorithm for the flattened butterfly (e.g. route in dimension1, then route in dimension 2); in this case, the routing algorithm itself is restrictive enough to prevent deadlock. Non-minimal routing allows the path diversity available in the flattened butterfly network to be used to improve load balance and performance. For these reasons, we use the non-minimal global adaptive routing (UGAL) [73] algorithm when evaluating the flattened butterfly topology in this work. UGAL routing balances load when needed by routing minimally to an



(a)



(b)

Figure 6.3: (a) Block diagram of a 2-dimension flattened butterfly consisting of 64 nodes. and (b) the corresponding layout of the flattened butterfly where dimension1 routers are horizontally placed and dimension2 router are vertically placed.

intermediate node in the first phase, and then routing minimally to the destination in the second phase. For such non-minimal routing algorithms, the number of VCs needed is $2d$, where d is the number of dimensions in the flattened butterfly. To reduce the number of VCs that are needed, we use DOR within each phase of UGAL routing – thus, only 2 VCs are needed.

6.4 Bypass Channels and Microarchitecture

As shown in Figure 6.3, the routers are fully connected in each dimension. Channels that pass over other routers in the same row or column function as *bypass channels.*

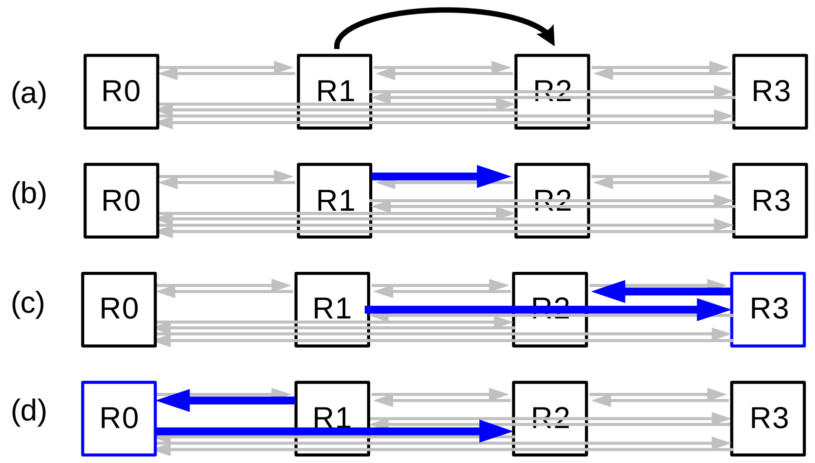
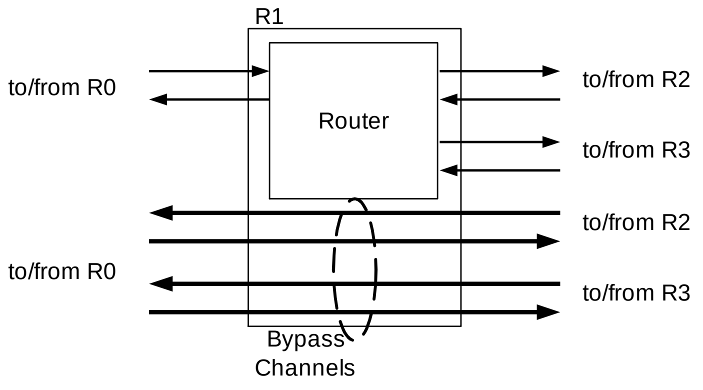
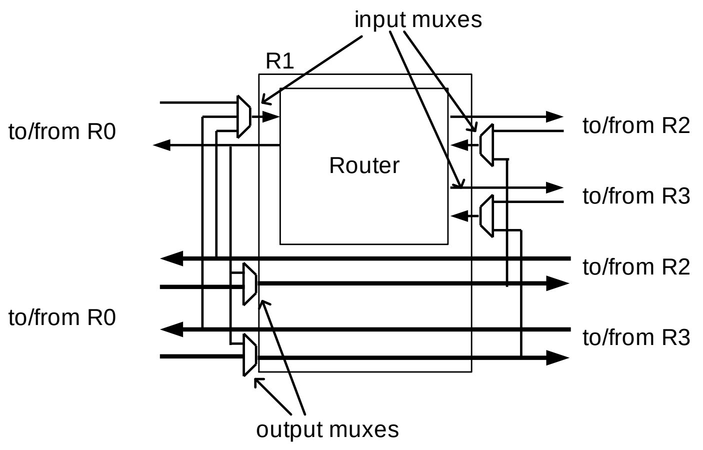


Figure 6.4: Routing paths in the 2D on-chip flattened butterfly. (a) All of the traffic from nodes attached to R1 is sent to nodes attached to R2. The minimal path routing is shown in (b) and the two non-minimal paths are shown in (c) and (d). For simplicity, the processing nodes attached to these routers are not shown.

Using non-minimal routing in the flattened butterfly topology increases both packet latency and energy consumption because packets are routed to intermediate routers for load-balancing purposes before being delivered to their destinations. The layout of an on-chip network can result in the non-minimal routes *overshooting* the destination on the way to the intermediate node selected for load-balancing, as shown in Figure 6.4(c). A non-minimal route may also route a packet away from its destination before it is routed back to its destination on a bypass channel that passes over the source (Figure 6.4(d)). To avoid the inefficiencies of routing packets on paths of non-minimal physical lengths, the bypass channels can be connected to those routers they pass over. These additional connections allow packets to enter or leave the bypass channels early when doing so is beneficial. In this section, we explain how the router microarchitecture and the flow control mechanisms can be modified to connect the bypass channels directly to the router switch in order to reduce latency and improve energy efficiency.



(a)



(b)

Figure 6.5: Flattened butterfly router diagram with bypass channels in a (a) conventional flattened butterfly router diagram and (b) flattened butterfly with muxes to efficiently use the bypass channels. The router diagram is illustrated for router R1 in Figure 6.4 with the connections shown for only single dimension of the flattened butterfly.

6.5 Router Bypass Architecture

A high-level diagram of a router in an on-chip flattened butterfly is shown in Figure 6.5(a). It consists of the switch and bypass channels that connect the neighbouring routers. One method to connect the bypass channels to the router is to add additional inputs to the switch. However, doing so would significantly increase the complexity of the switch. For example, in the flattened butterfly shown in Figure 6.3, the switch would increase from 10×10 to 18×18 in the worst case, nearly quadrupling the area consumed by the switch. In addition, the use of bypass channels is not intended to increase the bandwidth of the topology, but rather to reduce latency and energy – thus, the larger switch, which would provide additional bandwidth, is not needed. Instead, we break the bypass channels as they pass over the router and insert muxes as shown in Figure 6.5(b).

As illustrated in Figure 6.5(b), two types of muxes are added to connect the bypass channels to the local router: input muxes, and output muxes[2]. The inputs to the muxes can be classified as either bypass inputs (e.g. inputs from the bypass channels) or direct ports (e.g. inputs/outputs to/from the local router). The input muxes receive packets destined for the local router that would otherwise bypass the local router enroute to the intermediate node selected by the routing algorithm, as illustrated in Figure 6.4(c). Thus, each input mux receives both the direct inputs that the packet would have used if the non-minimal path was taken and the inputs from the bypass channels. The output muxes are used by packets that would have initially been routed away from their destinations before being routed back over the local router, as illustrated in Figure 6.4(d). The inputs to the output muxes are the direct outputs from the local router and the bypass channel inputs – the path the packet would have taken if non-minimal routing path was taken. The addition of these muxes does not eliminate the need for non-minimal routing for load-balancing purpose. Instead, the muxes reduce the distance traveled by packets to improve energy efficiency and reduce latency.

[2]Using the analogy of cars and highways, the long wires introduced correspond to adding highways. The input muxes correspond to adding additional exit ramps to the highways while the outputs muxes correspond to adding entrance ramps to get on the highway.

6.5.1 Mux Arbiter

The arbiters that control the bypass muxes are critical to the proper use of the bypass channels. A simple round-robin arbiter could be implemented at the muxes. While this type of arbiter leads to a locally fair arbitration at the mux, the arbiter does not guarantee global fairness. To provide global fairness, we implement a *yield* arbiter that yields to the *primary* input – i.e. the input that would have used the channel bandwidth at the output of the mux if the bypass channels were not connected to the local router. Accordingly, the direct input is given priority at an input mux, while the bypass channel is given priority at an output mux. Thus, if the primary input is idle, the arbiter opportunistically grants access to the non-primary input.

To prevent starvation of the non-primary inputs, a control packet is sent along the non-minimal path originally selected by the routing algorithm. This control packet contains only routing information and a marker bit to identify the packet as a control packet. The control packet is routed at the intermediate node as though it were a regular packet, which results in it eventually arriving at the primary input of the muxes at the destination router. When the control packet arrives, the non-primary input is granted access to the mux output bandwidth. This policy guarantees that a packet waiting at the non-primary input of a bypass mux will eventually be granted access to the bypass mux. In the worst-case (i.e. highly congested) environment, the latency of the non-minimal routed packets will be identical to the flattened butterfly that does not directly use the bypass channels. However, there will still be an energy savings because the flit does not traverse the non-minimal physical distance; instead, only the control flit, which is much smaller than a data packet, travels full physical distance of the non-minimal route.

6.5.2 Switch Architecture

With minimal routing, the crossbar switch can be simplified because it need not be fully connected. Non-minimal routing increases the complexity of the switch because some packets might need to be routed twice within a dimension, which requires more connections within the crossbar. However, by using the bypass channels efficiently,

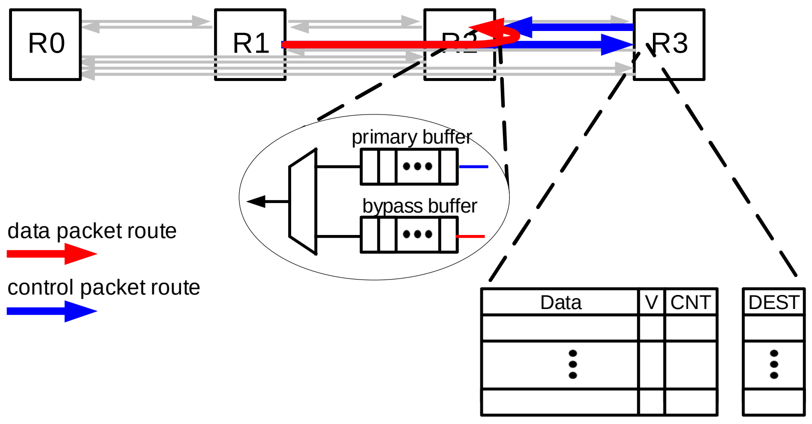


Figure 6.6: Modification to the buffers introduced into the flow control with the use of bypass channels. The additional bits of the buffers correspond to V : valid bit, CNT : count of control packet, and CTL corresponds to control packet content which contains a destination.

non-minimal routing can be implemented using a switch of lesser complexity – one that approaches the complexity of a flattened butterfly that only supports minimal routing. If the bypass channels are used, non-minimal routing does not require sending full packets through intermediate routers, and as a result, the connections within the switch themselves approaches that of the switch that only supports minimal routing.

6.5.3 Flow Control

Buffers are required at the non-primary inputs of the bypass muxes for flow control. This is illustrated in Figure 6.6. Thus with non-minimal routing, credits for the bypass buffers are needed before packet can depart a router. In addition, the minor modifications are required to the existing input buffers within the routers to correctly handle the control packets. These modification should introduce little overhead because the datapaths of on-chip networks are typically much wider than required for the control packet.

Topology	**Routing**
FBLY-MIN	randomized-dimension
FBLY-NONMIN	UGAL [73]
FBLY-BYPASS	UGAL [73]
CMESH	O1Turn with express channels [8]
MESH	O1Turn [70]

Table 6.1: Routing algorithms used in simulation comparison.

6.6 Evaluation

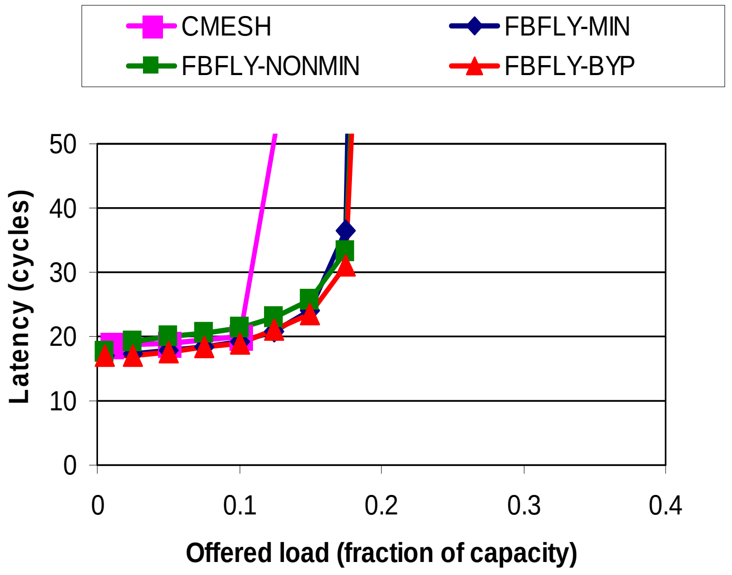
We compare the performance of the following topologies in this section:

1. conventional 2-D mesh (MESH)
2. concentrated mesh with express channels [8] (CMESH)
3. flattened butterfly (FBFLY)
   (a) flattened butterfly with minimal routing only (FBFLY-MIN)
   (b) flattened butterfly with non-minimal routing (FBFLY-NONMIN)
   (c) flattened butterfly with non-minimal routing and use of bypass channels (FBFLY-BYP)

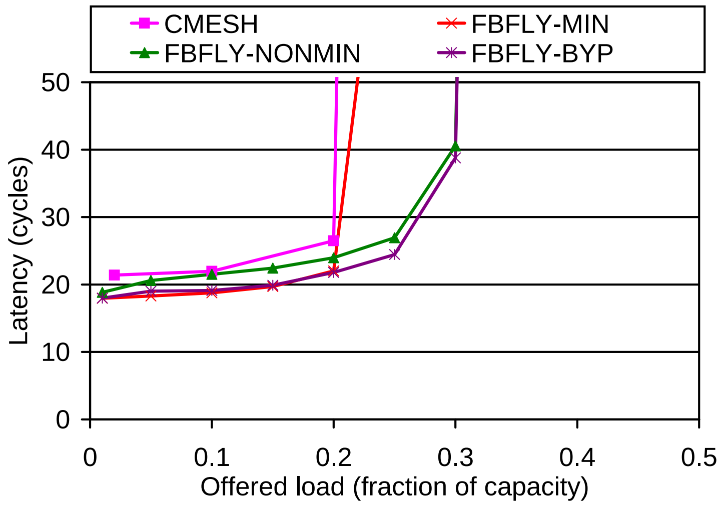
The topologies were evaluated using a cycle accurate network simulator. We compare the networks' performance and power consumption. The power consumption is based on the model described in [8] for a 65nm technology. We accurately model the additional pipeline delay required for the high-radix routers as well as the additional serialization latency through the narrower channels. The bisection bandwidth is held constant across all topologies for the purpose of comparing the performance of the different networks.

6.6.1 Performance

We compare the performance of the different topologies for a 64-node on-chip network by evaluating their throughput using open-loop simulation and also compare them



(a)



(b)

Figure 6.7: Throughput comparison of CMESH and FBFLY for (a) tornado and (b) bit complement traffic pattern.

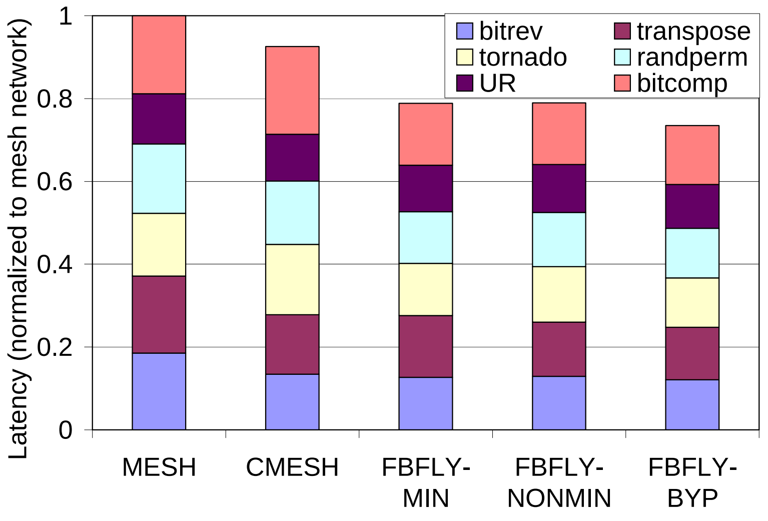


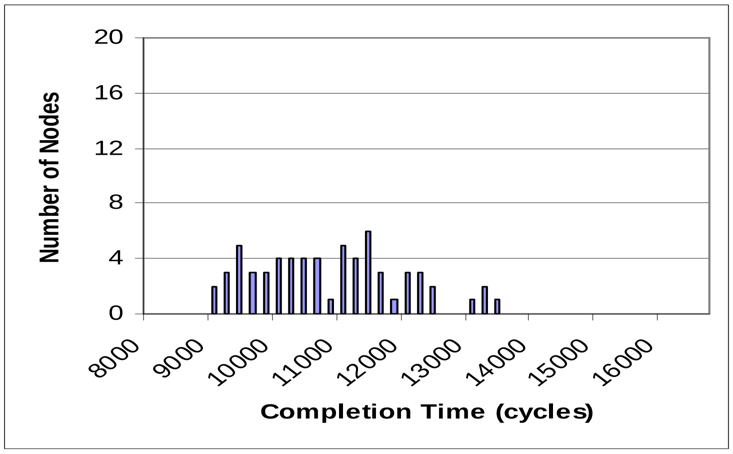
Figure 6.8: Latency comparison of alternative topologies across different synthetic traffic pattern.

using closed-loop simulation, using both synthetic traffic pattern and traces from simulations. The routing algorithms used for the different topologies are described in Table 6.1. For flow control, virtual channel flow control is used with 2 VCs to break routing deadlock and another 2 VCs needed to break protocol deadlock for the closed-loop simulations.

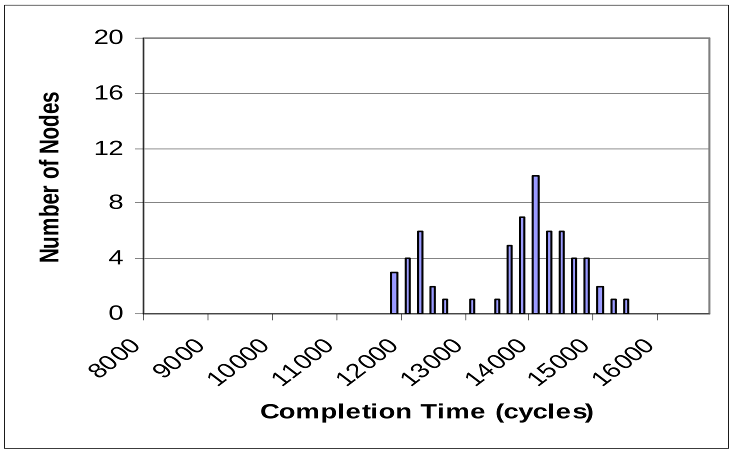
Throughput Comparison

To evaluate the throughput, the simulator is warmed up under load without taking measurements until steady-state is reached. Then a sample of injected packets is labeled during a measurement interval. The simulation is run until all labeled packets exit the system. For the throughput analysis, packets are assume to be single-flit packets.

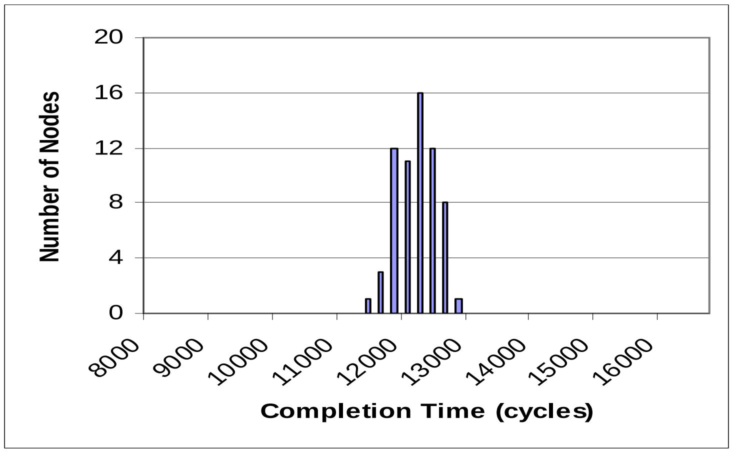
In Figure 6.7, we compare the latency vs. offered load on two adversarial traffic patterns for CMESH and FBFLY – two topologies that use concentration to reduce the cost of the network. By effectively exploiting non-minimal routing and smaller diameter of the topology, FBFLY can provide up to 50% increase in throughput



(a)



(b)



(c)

Figure 6.9: Node completion time variance for the different topologies (a) mesh (b) CMESH and (c) flattened butterfly.

compared to CMESH while provide lower zero-load latency. Although the MESH can provide higher throughput for some traffic pattern, it has been previously shown that CMESH results in a more cost- and energy-efficient topology compared to the MESH [8].

Synthetic Batch Traffic

In addition to the throughput measurement, we use a batch experiment to model the memory coherence traffic of a shared memory multiprocessor. Each processor executes a fixed number of remote memory operations (e.g. remote cache line read/write requests) during the simulation and we record the time required for all operations to complete. Read requests and write acknowledgements are mapped into 64-bit messages, while read replies and write requests are mapped into 576-bit messages. Each processor may have up to four outstanding memory operations. The synthetic traffic pattern used are uniform random (UR), bit complement, transpose, tornado, a random permutation, and bit reverse [22].

Figure 6.8 shows the performance comparison for the batch experiment and we normalize the latency to the mesh network. CMESH reduces latency, compared to the MESH, by 10% but the flattened butterfly reduces the latency further. By using FBFLY-NONMIN, the latency can actually *increase* because of the extra latency incurred with the non-minimal routing. However, the FBFLY-BYP provides the benefit of non-minimal routing but reducing the latency as all packets take minimal physical path and results in approximately 28% latency reduction, compared to the MESH network.

In addition to the latency required to complete the batch job, we also plot the variance of the completion time of the 64 nodes. A histogram is shown in Figure 6.9 that collects the completion time for *each* processing node. With the flattened butterfly, because of the lower diameter, the completion time has much more tighter distribution and smaller variance in the completion time across all of the nodes. Less variance can reduce the global synchronization time in chip multiprocessor systems. The CMESH, because it is not a symmetric topology, leads to an unbalanced distribution of completion time.

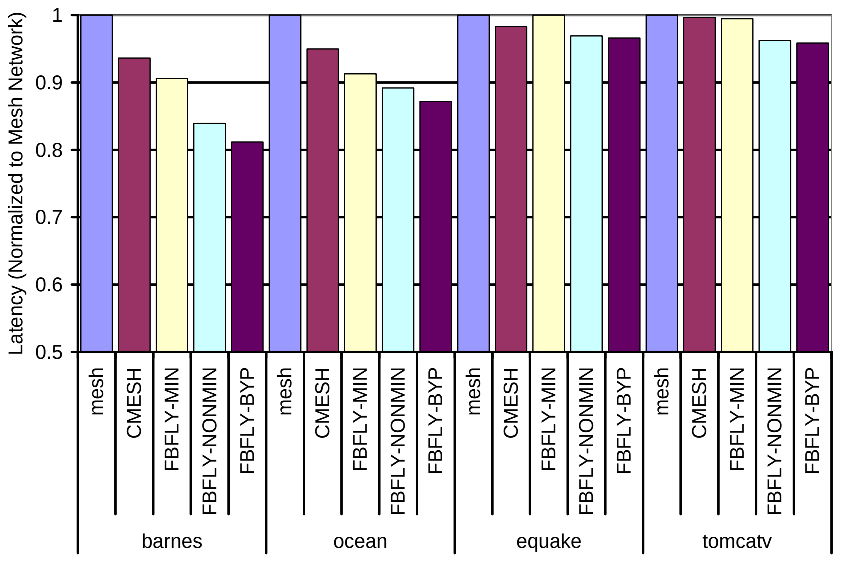


Figure 6.10: Performance comparison from SPLASH benchmark traces generated from a distributed TCC simulator.

Multiprocessor Traces

Network traces were collected from an instrumented version of a 64-processor directory based Transactional Coherence and Consistency (TCC) multiprocessor simulator [12]. In addition to capturing the traffic patterns and message distributions, we record detailed protocol information so that we can infer dependencies between messages and identify sequences of interdependent communication and computation phases at each processor node. This improves the accuracy of the performance measured when the traces are replayed through a cycle-accurate network simulator. When capturing the traces, we use an idealized interconnection network model which provides instantaneous message delivery to avoid adversely biasing the traces. The traffic injection process uses the recorded protocol information to reconstruct the state of each processor node. Essentially, each processor is modelled as being in one of two states: an active computing state, during which it progresses towards its next communication event; or, an idle communicating state, in which it is stalled waiting for an outstanding communication request to complete. Incoming communication events which do not require processor intervention, such as a remote read request, are assumed to be handled by the memory controller and therefore do not interrupt the receiving processor.

Four benchmarks (barnes, ocean, equake, and tomcatv) from the SPLASH benchmarks [81] were used to evaluate the alternative topologies and the results are shown in Figure 6.10. For two of the benchmarks (equake, tomcatv), the flattened butterfly on-chip network provides less than 5% reduction in latency. However, for the other two benchmarks (barnes, ocean), the flattened butterfly can provide up to 20% reduction in latency.

6.6.2 Power Comparison

The power consumption comparison is shown in Figure 6.11. The flattened butterfly provides additional power saving, compared to the CMESH. With the reduction in the width of the datapath, the power consumption of the crossbar is also reduced – thus, achieving approximately 38% power reduction compared to the mesh network.

The area of a router tends to increase with its radix. Control logic, such as the switch allocator, consumes area proportional to the square of the router radix; however, it represents a small fraction of the aggregate area. Consequently, the buffers and switch dominate the router area. The buffer area can be kept constant as the radix increases by reducing the buffer space allocated per input port. The total switch area can be approximated as $n(bk)^2$ where n is the number of routers, b is the bandwidth per port, and k is the router radix. As k increases, b decreases because the bisection bandwidth is held constant, and n also decreases, because each router services more processors. Consequently, we expect high-radix on-chip network will consume less area. We estimate that the flattened butterfly provides an area reduction of approximately 4x compared to the conventional mesh network and a reduction of 2.5x compared to the concentrated mesh. [3] Although the introduction of the bypass muxes can increase the area as well as the power, the impact is negligible compare to the area and power consumption of the buffers and the channels.

[3]Although there is a radix increase from radix-8 to radix-10 comparing CMESH to the flattened butterfly, since b is reduced in half, there is an overall decrease in total area.

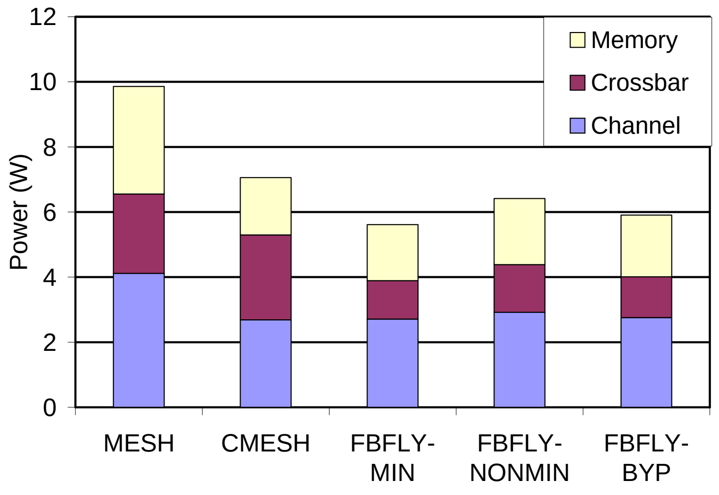


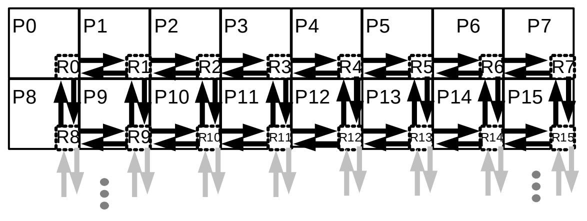
Figure 6.11: Power consumption comparison of alternative topologies on UR traffic.

6.7 Discussion

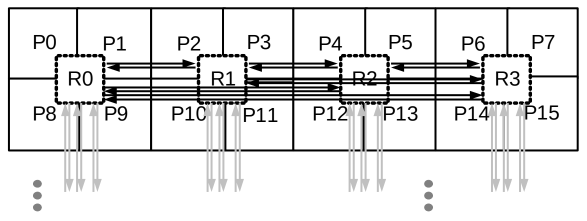
In this section we describe how the flattened butterfly topology can be scaled as more processors are integrated on chip. We also describe how the long direct links used in the flattened butterfly topology are likely to benefit from advances in on-chip signalling techniques, and how these direct links provide some of the performance benefits traditionally provided by virtual channels.

6.7.1 Comparison to Generalized Hypercube

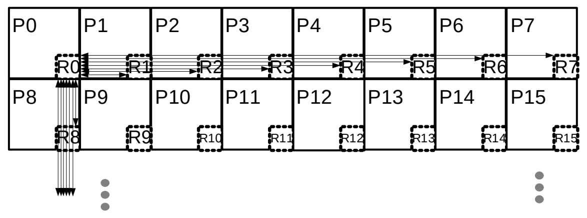
As described earlier in Section 3.6, the flattened butterfly topology is similar to the generalized hypercube [10] but the main difference is the use of concentration of in the flattened butterfly. The use of concentration in on-chip networks significantly reduces the wiring complexity because the resulting network requires fewer channels to connect the routers. Furthermore, it is often possible to provide wider channels when concentration is used, because there are fewer channels competing for limited wire resources, which improves the typical serialization latency. With the embedding of the topology into a planar VLSI layout constraint for on-chip networks, as the number of channels crossing a particular bisection increases, the total amount of



(a)



(b)



(c)

Figure 6.12: Layout of 64-node on-chip networks, illustrating the connections for the top two rows of nodes and routers for (a) a conventional 2-D mesh network, (b) 2-D flattened butterfly, and (c) a generalized hypercube. Because of the complexity, the channels connected to only R0 are shown for the generalized hypercube.

bandwidth needs to be divided among a larger number of channels – thus, decreasing the amount of bandwidth per channel.

A layout of a conventional mesh network and the 2-D flattened butterfly is shown in Figure 6.12(a,b). Although the flattened butterfly increases the number of channels crossing neighboring routers in the middle by a factor of 4, the use of concentration allows the two rows of wire bandwidth to be combined – thus, resulting in a reduction of bandwidth per channel by only a factor of 2. However, the generalized hypercube (Figure 6.12(c)) topology would increase the number of channels in the bisection of the network by a factor of 16, which would adversely impact the serialization latency and the overall latency of the network.

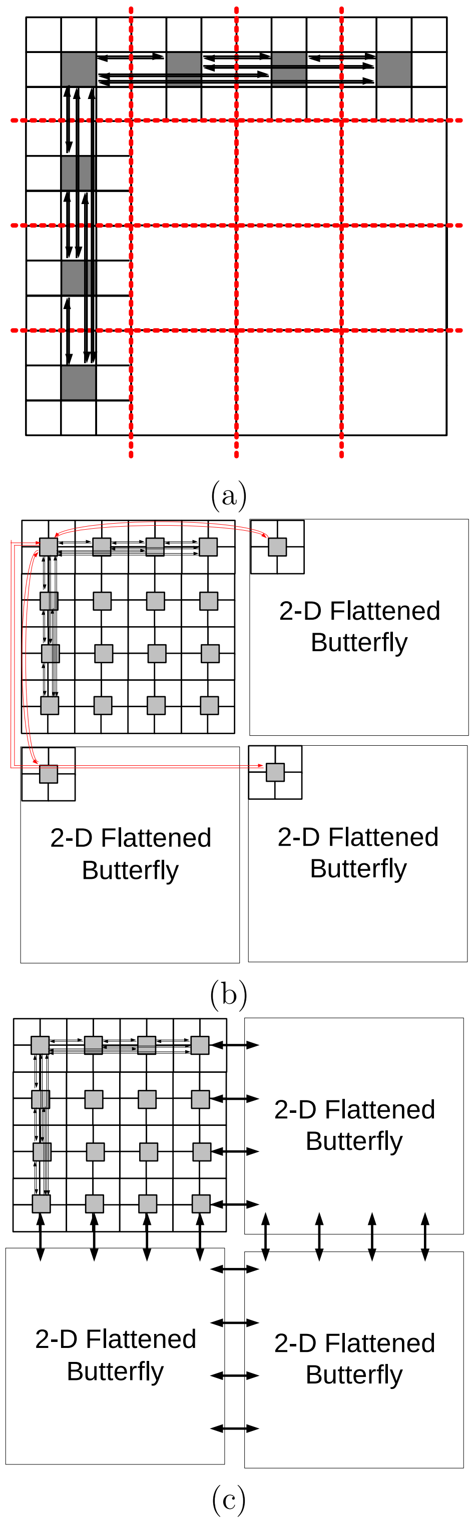


Figure 6.13: Different methods to scale the on-chip flattened butterfly by (a) increasing the concentration factor, (b) increasing the dimension of the flattened butterfly, and (c) using a hybrid approach to scaling.

6.7.2 Scaling On-Chip Flattened Butterfly

The flattened butterfly in on-chip networks can be scaled to accommodate more nodes in different ways. One method of scaling the topology is to increase the concentration factor. For example, the concentration factor can be increased from 4 to 8 to increase the number of nodes in the network from 64 to 128 as shown in Figure 6.13(a). This further increases the radix of the router to radix-14. With this approach, the bandwidth of the inter-router channels must to be properly adjusted such that there is sufficient bandwidth to support the terminal bandwidth.

Another scaling methodology is to increase the dimension of the flattened butterfly. For example, the dimension can be increased from a 2-D flattened butterfly to a 3-D flattened butterfly and provide an on-chip network with up to 256 nodes as shown in Figure 6.13(b). To scale a larger number of nodes, both the concentration factor as well as the number of dimensions can be increased as well.

However, as mentioned in Section 6.7.1, as the number of channels crossing the bisection increase, reduction in bandwidth per channel and increased serialization latency become problematic. To overcome this, hybrid approach can be used to scale the on-chip flattened butterfly. One such possibility is shown in Figure 6.13(c) where the 2-D flattened butterfly is used locally and the cluster of 2-D flattened butterfly is connected with a mesh network. This reduces the number channels crossing the bisection and minimizes the impact of narrower channels at the expense of slightly increase in the average hop count (compared to a pure flattened butterfly).

6.7.3 Future Technologies

The use of high-radix routers in on-chip networks introduces long wires. The analysis in this work assumed optimally repeated wires to mitigate the impact of long wires and used pipelined buffers for multicycle wire delays. However, many evolving technology will impact communication in future on-chip networks and the longer wires in the on-chip flattened butterfly topology are suitable to exploit these technologies. For example, on-chip optical signalling [46] and on-chip high-speed signalling [37] attempt

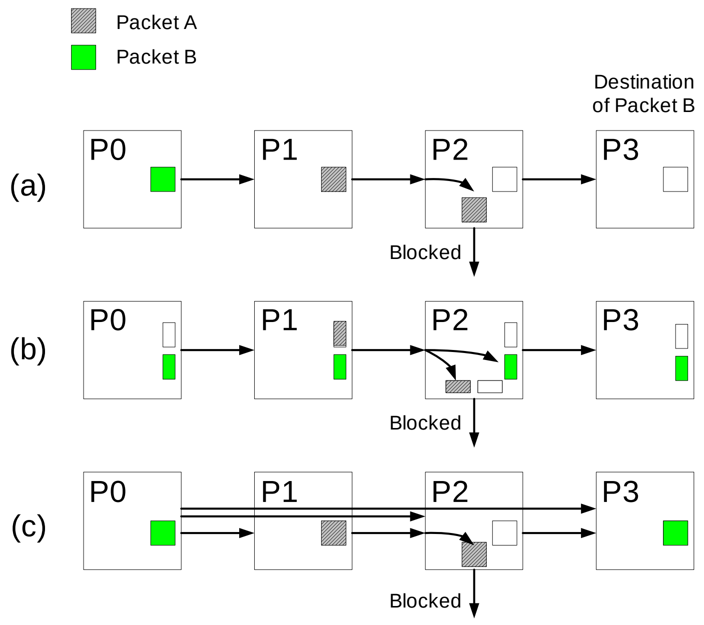


Figure 6.14: Block diagram of packet blocking in (a) wormhole flow control (b) virtual channel flow control and (c) flattened butterfly.

to provide signal propagation velocities that are close to the speed of light while providing higher bandwidth for on-chip communications. With a conventional 2D mesh topology, all of the wires are relatively short and because of the overhead involved in these technologies, low-radix topologies can not exploit their benefits. However, for the on-chip flattened butterfly topology which contains both short wires as well as long wires, the topology can take advantage of the cheap electrical wires for the short channels while exploiting these new technologies for the long channels.

6.7.4 Use of Virtual Channels

Virtual channels (VCs) were originally used to break deadlocks [21] and were also proposed to increase network performance by provide multiple virtual lanes for a single physical channel [23]. When multiple output VCs can be assigned to an input VC, a VC *allocation* is required which can significantly impact the latency and the area of a router. For example, in the TRIPS on-chip network, the VC allocation consumes 27% of the cycle time [31] and an area analysis show that VC allocation

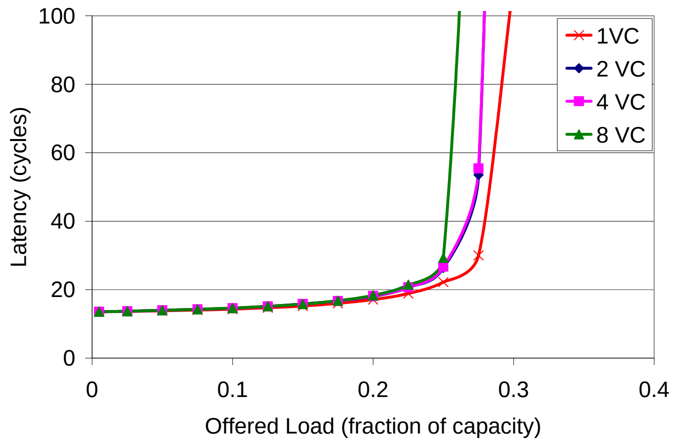


Figure 6.15: Performance comparison as the number of virtual channels is increased in the flattened butterfly.

can occupy up to 35% of the total area for an on-chip network router [59].

In Figure 6.14(a), an example of how blocking can occur in conventional network with wormhole flow control is shown. By using virtual channel flow control (Figure 6.14(b)), buffers are partitioned into multiple VCs, which allow packets to pass blocked packets. However, with the use of high-radix routers and the flattened butterfly topology, the additional wire resources available can be used to overcome the blocking that can occur as shown in Figure 6.14(c). As a result, blocking is reduced to only packets originating from the same source router and destined to the routers in the same column of the network. Thus, the topology reduces the need for the buffers to be partitioned into multiple VCs and takes advantage of the abundant wiring available in an on-chip network. VCs for other usage such as separating traffic into different classes might still be needed but such usage does not require VC allocation.

Figure 6.15 shows how the performance of the flattened butterfly is changed by increasing the number of VCs. In the simulation comparison, the total amount of storage per physical channel is held constant – thus, as the number of VCs is increased, the amount of buffering per VC is decreased. As a result, increasing the number of VCs for an on-chip flattened butterfly can slightly degrade the performance of the network since the amount of buffering per VC is reduced.

6.8 Related Work

On-chip interconnection networks have recently attracted considerable research attention [11], much of which has focused on microarchitecture improvements [57, 1] and routing algorithms [70]. Most on-chip networks that have been proposed are low-radix, mostly using a 2D mesh or a torus network [25]. Balfour and Dally proposed using concentrated mesh and express channels in on-chip networks to reduce the diameter and energy of the network [8]. This work expands on their idea and provides a symmetric topology that further reduces latency and energy. In addition, the benefits of creating parallel subnetworks [8] can also be applied to flattened butterfly topology in on-chip networks. Kumar et al. [48] proposed the use of express virtual channels (EVC) to reduce the latency of 2-D mesh on-chip network by bypassing intermediate routers. However, EVC is built on top of a 2-D mesh network and requires sharing the bandwidth between EVC and non-EVC packets and thus, does not completely eliminate intermediate routers.

The yield arbiter described in Section 6.5.1 is similar in concept to the flit-reservation flow control (FRFC) [62]. In FRFC, a control flit is sent ahead of the data flit and reserves the buffers and channels for the ensuing data flits. However, the scheme described in this paper uses the control flit to ensure bandwidth in the worst-case scenario – otherwise, the bypass channel is used regardless of the control flit.

The scaling of the topology with an hybrid (mesh) approach at the top level has been proposed for off-chip interconnection networks to reduce the length of the global cables [82]. Similar scaling can also be applied for on-chip networks with the benefits being not just shorter wires but also reduced on-chip wiring complexity.

6.9 Summary

In this chapter, we described how high-radix routers and the flattened butterfly topology can be mapped to on-chip networks to realize a more efficient on-chip network.

By reducing the number of routers and channels in the network, the flattened butterfly topology results in lower latency and lower power consumption compared to previously proposed on-chip network topologies. The 2-D flattened butterfly for a 64-node on-chip network achieves these benefits by reducing the number of intermediate routers such that with minimal routing, a packet traverses only 1 intermediate router from any source to any destination. In addition, we describe bypass channels in the topology that allows the use of non-minimal routing without traversing non-minimal physical distance – resulting in minimal increase in power while further reducing latency in the on-chip network. We show that the flattened butterfly can increase throughput by up to 50% compared to the concentrated mesh and reduce latency by 28% while reducing the power consumption by 38% compared to a mesh network.

Chapter 7

Conclusion and Future Work

The design of an interconnection network is significantly impacted by technology. As technology evolves, its impact on interconnection networks needs to re-evaluated. This thesis has shown that with the significant increase in pin-bandwidth, the additional bandwidth is best exploited by creating *high*-radix routers where the bandwidth is divided among a large number of ports instead of maintaining the number of ports small and increasing the bandwidth per port. By using high-radix routers to reduce the network diameter, both the latency and the cost of the network are reduced. The migration to high-radix networks impacts every aspect of an interconnection network. This thesis explores the topology, routing, and the microarchitecture aspect of high-radix interconnection networks as well as expanding the use of high-radix routers in on-chip networks.

Chapter 1 and Chapter 2 showed that with the exponential increase in pin-bandwidth over the past twenty years, the additional pin-bandwidth is more effectively utilized by increasing the radix or the degree of the routers to create high-radix networks. Chapter 3 proposed the flattened butterfly topology – a cost-efficient topology that can exploit high-radix routers. The flattened butterfly approaches the cost of a conventional butterfly network while providing the performance per cost of a folded-Clos network. Chapter 4 discusses routing on high-radix topologies and shows how transient load-imbalance can occur if adaptive routing is not properly implemented.

In addition, the use of global adaptive routing [73] is needed to fully exploit the benefits of the flattened butterfly topology. Conventional router microarchitecture does not scale to high radix, and Chapter 5 introduces a hierarchical crossbar organization that scales efficiently as the radix increases, and describes how it is implemented in the Cray YARC router. Finally, Chapter 6 examines how high-radix routers and the flattened butterfly topology can be extended to on-chip networks to realize similar benefits which include lower latency and lower cost.

7.1 Future Directions

The migration to high-radix interconnection networks has opened many opportunities for future research. This thesis has addressed the topology and the routing in high-radix networks but assumed the conventional virtual channel flow control [23] in simulations. With the network diameter significantly reduced with the use of high-radix routers, the appropriate flow control for high-radix networks is an interesting topic for future study.

In Chapter 3, the flattened butterfly topology was proposed as a cost-efficient topology for high-radix networks. However, it remains to be seen if there are more cost-efficient topologies or an *optimal* topology that exploits high-radix routers. One disadvantage of high-radix networks is the longer cables that are required compared to low-radix networks such as a 2-D or a 3-D torus. However, with the advent of economical optical signalling [36, 54], they enable topologies with long channels and mitigates the impact of channel length in the topology. The recently proposed dragonfly topology [44] is an example of a topology that exploits high-radix routers by creating *virtual* high-radix routers [1] and exploits the availability of optical active cables to minimize the number of long channels in the topology.

The hierarchical organization was proposed in Chapter 5 to provide a scalable router microarchitecture. However, as radix continues to increase, the hierarchical organization requires long wires to connect the subswitches in each row and each

[1]A virtual high-radix router is created by combining multiple high-radix router to further increase the effective radix of the network.

column. It remains to be seen if alternative switch microarchitectures, such as using a multistage network instead of a crossbar, can be used to provide a more scalable router microarchitecture.

Finally, the use of high-radix routers in on-chip networks leads to many interesting research directions. On-chip networks require low latency and low power. Thus, the design of on-chip high-radix routers needs to be carefully evaluated such that increasing the radix does not adversely impact the router clock cycle or the area. Compared to the conventional 2-D mesh network, the flattened butterfly was proposed as an efficient topology for on-chip networks in Chapter 6. However, as the number of on-chip terminals increase, it remains to be seen if the flattened butterfly topology can scale efficiently. Alternative topologies are worth exploring for future on-chip networks.

On-chip networks present different "packaging" constraints because the network needs to be embedded into a 2-D VLSI planar layout. [2] To exploit the 2-D VLSI planar layout, the concept of bypass channels was introduced in Chapter 6. The use of bypass channels further complicates the router design in two ways – the addition of bypass mux and additional flow control and non-uniform distribution of the bypass channels (i.e., the routers will have different number bypass channels). As a future work, it is interesting to explore generalizing the use of bypass channels to other topologies such that long channels can be exploited to achieve ideal latency in on-chip networks while minimizing physical distance traversed for packets that do not need to traverse the entire long channel.

[2]The use of 3-D integration changes this constraint but the on-chip network within each layer still needs to be packaged in a 2-D VLSI planar layout.

Bibliography

[1] Pablo Abad, Valentin Puente, Jose Angel Gregorio, and Pablo Prieto. Rotary router: An efficient architecture for cmp interconnection networks. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 116–125, San Diego, CA, June 2007.

[2] Dennis Abts, Abdulla Bataineh, Steve Scott, Greg Faanes, James Schwarzmeier, Eric Lundberg, Tim Johnson, Mike Bye, and Gerald Schwoerer. The Cray Black-Widow: A Highly Scalable Vector Multiprocessor. In *Proceedings of the International Conference for High-Performance Computing, Network, Storage, and Analysis (SC'07)*, Reno, NV, November 2007.

[3] A. Agarwal. Limits on Interconnection Network Performance. *IEEE Trans. Parallel Distrib. Syst.*, 2(4):398–412, 1991.

[4] A. Agarwal, R. Bianchini, D. Chaiken, K. Johnson, D. Kranz, J. Kubiatowicz, B-H. Lim, K. Mackenzie, and D. Yeung. The MIT Alewife Machine: Architecture and Performance. In *Proc. of the 22nd Annual Int'l Symp. on Computer Architecture*, pages 2–13, 1995.

[5] Anant Agarwal, Liewei Bao, John Brown, Bruce Edwards, Matt Mattina, Chyi-Chang Miao, Carl Ramey, and David Wentzlaff. Tile Processor: Embedded Multicore for Networking and Multimedia. In *Hot Chips 19*, Stanford, CA, Aug. 2007.

[6] Amphenol. http://www.amphenol.com/.

[7] Yucel Aydogan, Craig B. Stunkel, Cevdet Aykanat, and Bülent Abali. Adaptive source routing in multistage interconnection networks. In *IPPS '96: Proceedings of the 10th International Parallel Processing Symposium*, pages 258–267, Honolulu, HW, 1996. IEEE Computer Society.

[8] James Balfour and William J. Dally. Design tradeoffs for tiled cmp on-chip networks. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, pages 187–198, 2006.

[9] Jon Beecroft, David Addison, Fabrizio Petrini, and Moray McLaren. Quadrics QsNet II: A Network for Supercomputing Applications. In *Hot Chips 15*, Stanford, CA, August 2003.

[10] Laxmi N. Bhuyan and Dharma P. Agrawal. Generalized hypercube and hyperbus structures for a computer network. *IEEE Trans. Computers*, 33(4):323–333, 1984.

[11] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, 38(1):1, 2006.

[12] Hassan Chafi, Jared Casper, Brian D. Carlstrom, Austen McDonald, Chi Cao Minh, Woongki Baek, Christos Kozyrakis, and Kunle Olukotun. A scalable, non-blocking approach to transactional memory. In *13th International Symposium on High Performance Computer Architecture (HPCA)*. Feb 2007.

[13] Kun-Yung Ken Chang, Jason Wei, Charlie Huang, Simon Li, Kevin Donnelly, Mark Horowitz, Yingxuan Li, and Stefanos Sidiropoulos. A 0.4–4-Gb/s CMOS Quad Transceiver Cell Using On-Chip Regulated Dual-Loop PLLs. *IEEE Journal of Solid-State Circuits*, 38(5):747–754, 2003.

[14] Andrew A. Chien. A cost and speed model for k-ary n-cube wormhole routers. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):150–162, 1998.

[15] Sangyeun Cho and Lei Jin. Managing distributed, shared l2 caches through os-level page allocation. In *Annual IEEE/ACM International Symposium on Microarchitecture*, pages 455–468, Orlando, FL, 2006.

[16] Yungho Choi and Timothy Mark Pinkston. Evaluation of Crossbar Architectures for Deadlock Recovery Routers. *J. Parallel Distrib. Comput.*, 61(1):49–78, 2001.

[17] C Clos. A Study of Non-Blocking Switching Networks. *The Bell System technical Journal*, 32(2):406–424, March 1953.

[18] Cray X1. http://www.cray.com/products/systems/x1/.

[19] W. J. Dally. Performance Analysis of k-ary n-cube Interconnection Networks. *IEEE Transactions on Computers*, 39(6):775–785, 1990.

[20] W. J. Dally, P. P. Carvey, and L. R. Dennison. The Avici Terabit Switch/Router. In *Proc. of Hot Interconnects*, pages 41–50, August 1998.

[21] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, 36(5):547–553, 1987.

[22] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, San Francisco, CA, 2004.

[23] William J. Dally. Virtual-channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, 1992.

[24] William J. Dally and Charles L. Seitz. The Torus Routing Chip. *Distributed Computing*, 1(4):187–196, 1986.

[25] William J. Dally and Brian Towles. Route packets, not wires: on-chip inteconnection networks. In *Proc. of the 38th conference on Design Automation (DAC)*, pages 684–689, 2001.

[26] H. G. Dietz and T.I.Mattox. Compiler techniques for flat neighborhood networks. In *13th International Workshop on Languages and Compilers for Parallel Computing*, pages 420–431, Yorktown Heights, New York, 2000.

[27] T. Dunigan. Early Experiences and Performance of the Intel Paragon. *Technical report, Oak Ridge National Laboratory, ORNL/TM-12194.*, 1993.

[28] A. P. J. Engbersen. Prizma Switch Technology. *IBM J. Res. Dev.*, 47(2-3):195–209, 2003.

[29] Kourosh Gharachorloo, Madhu Sharma, Simon Steely, and Stephen Van Doren. Architecture and Design of AlphaServer GS320. In *Proc. of the 9th Int'l conf. on Architectural support for programming languages and operating systems*, pages 13–24, 2000.

[30] Gore. http://www.gore.com/electronics.

[31] P. Gratz, C. Kim, R. McDonald, S.W. Keckler, and D.C. Burger. Implementation and Evaluation of On-Chip Network Architectures. In *International Conference on Computer Design (ICCD)*, San Jose, CA, 2006.

[32] Fred Heaton, Bill Dally, Wayne Dettloff, John Eyles, Trey Greer, John Poulton, Teva Stone, and Steve Tell. A Single-Chip Terabit Switch. In *Hot Chips 13*, Stanford, CA, 2001.

[33] Steven Heller. Congestion-Free Routing on the CM-5 Data Router. In *Parallel Computer Routing and Communication Workshop*, pages 176–184, Seattle, WA, 1994.

[34] Mark Horowitz, Chih-Kong Ken Yang, and Stefanos Sidiropoulos. High-Speed Electrical Signaling: Overview and Limitations. *IEEE Micro*, 18(1):12–24, 1998.

[35] IBM Redbooks. *An Introduction to the New IBM eServer pSeries High Performance Switch*.

[36] Intel Connects Cables. http://www.intel.com/design/network/products/optical/cables/index.h

[37] A.P. Jose, G. Patounakis, and K.L. Shepard. Near speed-of-light on-chip interconnects using pulsed current-mode signalling. In *Digest of Technical Papers. 2005 Symposium on VLSI Circuits*, pages 108–111, 2005.

[38] J. Jun, S. Byun, B. Ahn, Seung Y. Nam, , and D. Sung. Two-Dimensional Crossbar Matrix Switch Architecture. In *Asia Pacific Conf. on Communications*, pages 411–415, September 2002.

[39] M. J. Karol, M. G. Hluchyj, and S. P. Morgan. Input versus Output Queueing on a Space-division Packet Switch. *IEEE Transactions on Communications*, COM-35(12):1347–1356, 1987.

[40] John Kim, James Balfour, and William J. Dally. Flattened butterfly topology for on-chip networks. *IEEE Computer Architecture Letters*, 6(1), 2007.

[41] John Kim, James Balfour, and William J. Dally. Flattened butterfly topology for on-chip networks. In *Annual IEEE/ACM International Symposium on Microarchitecture*, pages 172–182, Chicago, IL, December 2007.

[42] John Kim, William J. Dally, and Dennis Abts. Adaptive Routing in High-radix Clos Network. In *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'06)*, Tampa, FL, 2006.

[43] John Kim, William J. Dally, and Dennis Abts. Flattened Butterfly : A Cost-Efficient Topology for High-Radix Networks. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 126–137, San Diego, CA, June 2007.

[44] John Kim, William J. Dally, Steve Scott, and Dennis Abts. Technology-Driven Highly-Scalable Dragonfly Topology. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, Beijing, China, June 2008.

[45] John Kim, William J. Dally, Brian Towles, and Amit K. Gupta. Microarchitecture of a high-radix router. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 420–431, Madison, WI, 2005.

[46] Nevin Kirman, Meyrem Kirman, Rajeev K. Dokania, Jose F. Martinez, Alyssa B. Apsel, Matthew A. Watkins, and David H. Albonesi. Leveraging optical technology in future bus-based chip multiprocessors. In *Annual IEEE/ACM International Symposium on Microarchitecture*, pages 492–503, Orlando, FL, 2006.

[47] Clyde P. Kruskal and Marc Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Trans. Computers*, 32(12):1091–1098, 1983.

[48] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jhay. Express virtual channels: Towards the ideal interconnection fabric. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 150–161, San Diego, CA, June 2007.

[49] Richard E. Ladner and Michael J. Fischer. Parallel prefix computation. *J. ACM*, 27(4):831–838, 1980.

[50] James Laudon and Daniel Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proc. of the 24th Annual Int'l Symp. on Computer Architecture*, pages 241–251, 1997.

[51] M J Edward Lee, William J. Dally, Ramin Farjad-Rad, Hiok-Tiaq Ng, Ramesh Senthinathan, John H. Edmondson, and John Poulton. CMOS High-Speed I/Os - Present and Future. In *International Conf. on Computer Design*, pages 454–461, San Jose, CA, 2003.

[52] C. Leiserson. Fat-trees: Universal networks for hardware efficient supercomputing. *IEEE Transactions on Computer*, C-34(10):892–901, October 1985.

[53] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Carl R. Feynman, Mahesh N. Ganmukhi, Jeffrey V. Hill, W. Daniel Hillis, Bradley C. Kuszmaul, Margaret A. St Pierre, David S. Wells, Monica C. Wong-Chan, Shaw-Wen Yang, and Robert Zak. The Network Architecture of the Connection Machine CM-5. *J. Parallel Distrib. Comput.*, 33(2):145–158, 1996.

[54] Luxtera Blazar LUX5010. http://www.luxtera.com/products_blazar.htm.

[55] Microprocessor Report. http://www.mdronline.com/.

[56] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 network architecture. In *Hot Chips 9*, pages 113–117, Stanford, CA, August 2001.

[57] Robert D. Mullins, Andrew West, and Simon W. Moore. Low-latency virtual-channel routers for on-chip networks. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 188–197, Munich, Germany, 2004.

[58] Myrinet. http://www.myricom.com/myrinet/overview/.

[59] Chrysostomos A. Nicopoulos, Dongkook Park, Jongman Kim, N. Vijaykrishnan, Mazin S. Yousif, and Chita R. Das. Vichar: A dynamic virtual channel regulator for network-on-chip routers. In *Annual IEEE/ACM International Symposium on Microarchitecture*, Orlando, FL, 2006.

[60] Michael D. Noakes, Deborah A. Wallach, and William J. Dally. The J-Machine Multicomputer: An Architectural Evaluation. In *Proc. of the 20th Annual Int'l Symp. on Computer Architecture*, pages 224–235, 1993.

[61] Greg Pautsch. Thermal Challenges in the Next Generation of Supercomputers. *CoolCon*, 2005.

[62] Li-Shiuan Peh and William J. Dally. Flit-reservation flow control. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 73–84, 2000.

[63] Li Shiuan Peh and William J. Dally. A Delay Model for Router Microarchitectures. *IEEE Micro*, 21(1):26–34, 2001.

[64] Fabrizio Petrini, Wuchun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics Network: High-Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, 2002.

[65] G. Pfister. *An Introduction to the InfiniBand Architecture (http://www.infinibandta.org)*. IEEE Press, 2001.

[66] R. Rojas-Cessa, E. Oki, and H. Chao. CIXOB-k: Combined Input-Crosspoint-Output Buffered Packet Switch. In *Proc. of IEEE Global Telecommunications Conf.*, pages 2654–2660, San Antonio, TX, 2001.

[67] S. Scott and G. Thorson. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. In *Hot Chips 4*, Stanford, CA, Aug. 1996.

[68] Steve Scott, Dennis Abts, John Kim, and William J. Dally. The BlackWidow High-radix Clos Network. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 16–28, Boston, MA, June 2006.

[69] Steven L. Scott and James R. Goodman. The impact of pipelined channels on k-ary n-cube networks. 5(1):2–16, 1994.

[70] Daeho Seo, Akif Ali, Won-Taek Lim, Nauman Rafique, and Mithuna Thottethodi. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 432–443, Madison, WI, 2005.

[71] SGI Altix 3000. http://www.sgi.com/products/servers/altix/.

[72] Howard Jay Siegel. A model of simd machines and a comparison of various interconnection networks. *IEEE Trans. Computers*, 28(12):907–917, 1979.

[73] Arjun Singh. *Load-Balanced Routing in Interconnection Networks.* PhD thesis, Stanford University, 2005.

[74] Arjun Singh, William J. Dally, Amit K. Gupta, and Brian Towles. GOAL: A load-balanced adaptive routing algorithm for torus networks. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 194–205, San Diego, CA, June 2003.

[75] C. B. Stunkel, D. G. Shea, B. Aball, M. G. Atkins, C. A. Bender, D. G. Grice, P. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker. The SP2 High-performance Switch. *IBM Syst. J.*, 34(2):185–204, 1995.

[76] Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 341–353, Anaheim, California, 2003.

[77] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, 1982.

[78] Sriram Vangal, Jason Howard, Gregory Ruhl, Saurabh Dighe, Howard Wilson, James Tschanz, David Finan, Priya Iyer, Arvind Singh, Tiju Jacob, Shailendra Jain, Sriram Venkataraman, Yatin Hoskote, and Nitin Borkar1. An 80-Tile 1.28 TFLOPS Network-on-Chip in 65nm CMOS. In *2007 IEEE Int'l Solid-State Circuits Conf., Digest of Tech. Papers (ISSCC 07)*, 2007.

[79] Hangsheng Wang, Li Shiuan Peh, and Sharad Malik. Power-driven Design of Router Microarchitectures in On-chip Networks. In *Proc. of the 36th Annual IEEE/ACM Int'l Symposium on Microarchitecture*, pages 105–116, 2003.

[80] Koon-Lun Jackie Wong, Hamid Hatamkhani, Mozhgan Mansuri, and Chih-Kong Ken Yang. A 27-mW 3.6-Gb/s I/O Transceiver. *IEEE Journal of Solid-State Circuits*, 39(4):602–612, 2004.

[81] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 24–36, Santa Margherita Ligure, Italy, 1995.

[82] Michale Woodacre. Towards multi-paradigm computing at sgi. Talk at Stanford University EE380 seminar, January 2006.

[83] S.D. Young and S Yalamanchili. Adaptive routing in generalized hypercube architectures. In *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*, pages 564–571, Dallas, TX, December 1991.