Polymorphic On-Chip Networks

Martha Mercaldi Kim*, John D. Davis†, Mark Oskin*, Todd Austin‡

*University of Washington
Computer Science and Engineering
Seattle, WA 98195
{mercaldi,oskin}@cs.washington.edu

†Microsoft Research, Silicon Valley
Mountain View, CA 94043
john.d@microsoft.com

‡University of Michigan
Electrical Engineering and Computer Science
Ann Arbor, MI 48109
austin@umich.edu

Abstract

As the number of cores per die increases, be they processors, memory blocks, or custom accelerators, the on-chip interconnect the cores use to communicate gains importance. We begin this study with an area-performance analysis of the interconnect design space. We find that there is no single network design that yields optimal performance across a range of traffic patterns. This indicates that there is an opportunity to gain performance by customizing the interconnect to a particular application or workload.

We propose polymorphic on-chip networks to enable per-application network customization. This network can be configured prior to application runtime, to have the topology and buffering of arbitrary network designs. This paper proposes one such polymorphic network architecture. We demonstrate its modes of configurability, and evaluate the polymorphic network architecture design space, producing polymorphic fabrics that minimize the network area overhead. Finally, we expand the network on chip design space to include a polymorphic network design, showing that a single polymorphic network is capable of implementing all of the pareto optimal fixed-network designs.

1 Introduction

On-chip networks play a critical role in the performance of computing systems, from high-speed network routers, to embedded devices, to chip multiprocessors (CMPs). Moving forward, as we integrate progressively more functionality on a single die, the communication infrastructure that binds them will play a central role in overall chip performance. Foreseeing this, researchers have developed innovations in all aspects of network-on-chip (NoC) design, including novel topologies [17, 5], routing algorithms [1], and switches optimized for latency [19, 21], fault tolerance [4], and power consumption [25].

We motivate our work by showing, through simulations of over 1500 design points, that no single network provides optimal performance across a range of traffic patterns. For example, a common network benchmark, uniformly random all-to-all communication, performs best with a network supporting direct non-local links, such as a fat tree; while a streaming-style benchmark is well suited for a simple mesh or ring network with efficient nearest neighbor communication support. Furthermore, even when optimal designs share a topology, the networks are provisioned very differently in terms of buffering and packet sizes. If an application is fixed, for example in an embedded device, its communication pattern is likely also to be fixed, and the network can be tailored to that traffic without penalty. When the application and traffic pattern varies, as one finds in programmable devices such as CMPs, overall performance suffers.

To address this problem, we introduce the *polymorphic on-chip network*. The polymorphic network consists of a configurable collection of network building blocks, principally queues and crossbars, in which an arbitrary network can be instantiated. The polymorphic network supports customization of network topology, link width, and buffering, to be determined post-fabrication, but prior to application runtime, thereby affording the opportunity to customize the on-chip network to each application. Prior work [13, 12, 22] indicates that there is significant opportunity to improve performance by tailoring the network on a per-application basis. Until now, however, it has not been possible to realize these benefits with a single hardware configuration.

We demonstrate how to configure the polymorphic network architecture to customize network topology, link bandwidth, and buffer capacity. The network pays for this flexibility with additional area overhead. We explore the polymorphic network design space by instantiating the networks composing the initial network-on-chip design space in each candidate polymorphic design. We identify the polymorphic network designs that incur the least area overhead relative to fixed hardware implementations. The most efficient in-

stances of the architecture we propose incur a 40% increase in interconnect area, which, for an interconnect that comprises 15% of a die, represents 6% of total die area.

In exchange for this extra area, the polymorphic network affords design flexibility. For a fixed area budget, our studies indicate that if you spent that area on a polymorphic network you could implement a large percentage – ranging from 80% all the way up to 97% – of the fixed-function networks that you could have built. Furthermore, the polymorphic network can emulate all of the previously-identified, optimal network designs. In summary, this paper makes the following contributions:

- We present a detailed design space exploration and Pareto analysis of NoC architectures.
- We introduce the polymorphic on-chip network, and demonstrate how to configure it into a variety of network topologies, buffering requirements, and link bandwidths.
- We explore the architectural design space of the polymorphic network, identifying the most area-efficient members.
- We extend the original NoC design space to include the polymorphic network, demonstrating the improvement in overall performance achievable with the a polymorphic network.

In the next section we begin by providing background material on NoCs. The reader highly familiar with NoC work may skip this and proceed directly to Section 3 which contains the NoC design space exploration and Pareto analysis. In Section 4, we describe the polymorphic network architecture and demonstrate how to use it, and then in Section 5, we explore the polymorphic network design space and expand the original space of network designs to include polymorphic networks.

2 Background

On-chip networks are a rich research domain, covering both breadth and depth. In the comparatively narrow space of this section, we outline the research context most necessary to frame the content of this paper.

Network designs: The history of on-chip networks begins with off-chip, large-scale system interconnects. This previous art used a variety of topologies, amongst them the ones we will include in our study: fat tree [20], butterfly [6], mesh [6] and ring [6]. Present and future multi-core designs demand much more than a simple integration of earlier large-scale system interconnects onto a single die. For example, while mapping meshes and rings is relatively easy, mapping a butterfly network onto a single chip presents more challenges [17]. In addition, new topologies, specifically designed for on-chip interconnects have arisen [17, 5].

Meanwhile, the switches that are connected to form these topologies have also been refined, to meet the stringent power, latency, and fault tolerance requirements of an on-chip network [4, 19, 25, 21]. The result is that today, we have a rich and expanding array of options when implementing an on-chip interconnect. Despite this array of choices, there is contemporary evidence that tailoring an interconnect to a particular workload can frequently and significantly improve performance.

Tailored NoC designs: There are a number of published techniques for determining the appropriate network design for a specific application. In 2006, Hu et al. [13] presented evidence that non-uniform network input buffers offer significant network performance improvements. The work then presents a pre-fabrication, buffer allocation algorithm that can be applied to find the appropriate buffer sizings to improve performance and economize resources. Other researcher aims to develop tools to synthesize a custom network on chip for a particular application [12, 22]. Each of these projects has demonstrated a significant improvement in performance when the interconnect was tailored to a particular application.

Relationship to FPGAs: Field-programmable gate arrays (FPGAs) already contain configurable networks. While certainly the concept of "island style" routing and FPGAs have inspired our work, the details and underlying purpose are entirely different. The performance (delay, power, area) of FPGAs is impacted by their design goals. The ability to configure routes on a per-bit level, means the overhead is quite high. Consequently, a long history of research into coarse-grained FPGAs exists [24, 11, 2]. This work reduces the overhead caused by bit-level configurability, but still provides interconnect structures geared towards fully statically routed designs, as would be required for emulating circuits. As will be evident later, the relationship between our polymorphic on-chip network architecture and an FPGA is only at a superficial level. In effect, a polymorphic network is a tailored configurable device designed for emulating on-chip networks, in the same way that an FPGA is a tailored configurable device designed for emulating circuits.

3 NoC Design Space Exploration

We begin our study by exploring the NoC design space, examining the performance of a range of network architectures operating under different traffic patterns. The first part of this section outlines the methodology we employ for this study, and the second part presents our results.

3.1 Methodology

NoC Designs: To explore a wide range of network on chip (NoC) designs, we vary not only the network parameters,

such as queue capacities and packet sizes, but also the policies and structures of the network, via the topology and arbitration algorithm. The first section of Table 1 enumerates the specific parameters and parameter values that define the design space.

We selected the topologies to cover a range of degrees of connectivity – from the highly connected fat tree, to the less connected ring – and because they belong to several widely-used topology families. For each topology, we used a deterministic, minimal, oblivious source routing algorithm. The design space includes two buffered arbitration policies: *store-and-forward* and *wormhole* [6] which reserves and preserves a connection at each switch until all packets in a packet have traversed the switch.

The network is defined by the topology, routing algorithm and arbitration policy. Within this definition, however, the performance of a network can vary a great deal based on its resources. Thus we also include a range of queue capacities and packet sizes, as outlined in Table 1, in the design space.

Cycle-level Simulation: We measure the performance of each NoC in the design space using a microarchitectural network simulator. The simulator is execution-driven and models the network on a cycle-by-cycle basis. We validated this simulator by successfully reproducing data from prior art [25].

Traffic Patterns: To drive the simulations, we use three synthetic workloads. The first of these, *uniform random* traffic, is a widely used traffic generation pattern, in which each node generates a message in each cycle for another randomly chosen node. This workload has similar traffic characteristics to an application that has near-random data accesses running on a CMP, for example a breadth first graph traversal [26]. For this workload, and the others used in this study, the packets are injected via a Bernoulli process.

The other two workloads are permutation workloads, in which each node sends packets to exactly one other node. This is sometimes called an adversarial pattern because, unlike in the uniform random pattern, the communication load is unbalanced. We experiment with two workloads in this family: *random permutation*, in which the permutation of sender to receiver nodes is randomly generated, and *nearest neighbor*, in which each sender sends packets to a nearby receiver. The local adversarial pattern reasonably approximates a system-on-chip design in which the designer took care to place communicating blocks near to one another to maximally exploit any locality in the application.

Area Model: As on-chip interconnects are often designed under tight area budgets, it is unrealistic to explore interconnection options on the basis of performance alone. Thus, we adopt the methodology of previous work [3], and develop an analytical area model for a network design. To balance model accuracy with design space size, we synthesize designs for the base network components, including buffers, queues and crossbars. These base estimates target a 90nm process, using high-performance GT standard cell libraries from Taiwan Semiconductor Manufacturing Company (TSMC) for memories, and models for full-custom layout of crossbar interconnects.

Table 1 describes the model in more detail. The inputs to the model in the first section of the table are the parameter settings which define a particular network design. The synthesis-based section of the table, specifies the base area components, including queue and crossbar areas. We then analytically combines the areas of the base components to estimate the total network circuit area. Because networks are wiring-heavy circuits, we conservatively assume a 20% wiring overhead [3] in the cell placement, in addition to the already-accounted-for crossbar.

3.2 Design Space Exploration Results

Fixed Area Budget: We found that previous work has allocated a remarkably consistent 15% of die area to on-chip interconnects [10, 18, 14]. These estimates span a wide range of process technologies, from 130nm all the way down to 32nm, and core counts, from eight to sixty-four core designs. We allocate 15% of a 15x15mm die ($225\mathrm{mm}^2$) to the interconnect, and subject to this area constraint investigate the performance tradeoffs in network latency and throughput.

The graphs in Figure 1 come from 64-node ($N = 64$) network simulations. On the x-axis is average packet latency while on the y is the average throughput. There is one graph per traffic pattern, with each point in the graphs representing one network design. The circled points indicate the designs that are pareto optimal for the given workload. These are the designs for which one dimension cannot be improved (latency reduced or througput increased) without sacrificing in the other dimension (increasing latency or reducing throughput).

First, we can examine the design tradeoffs when implementing a network for a single traffic pattern. For example, the first graph shows that the pareto optimal network designs for uniform random traffic include all four topologies. The fat tree and mesh offer the lowest latency optimal designs, followed by the butterfly with double the latency, and lastly by the ring with quadruple. Over these optimal designs, increases in throughput come only with accompanying increases in latency. With both the fat tree and mesh, short queues, of four entries, and large, 128-bit packets give rise to the aggregate optimal performance. At the other end of the spectrum, the ring, when provisioned with the same large packets, has a long transmission latency, but offers significantly higher throughput.

In the case of random adversarial traffic, Figure 1 indicates that fat tree once again offers the lowest-latency communication option. As with uniform random communica-

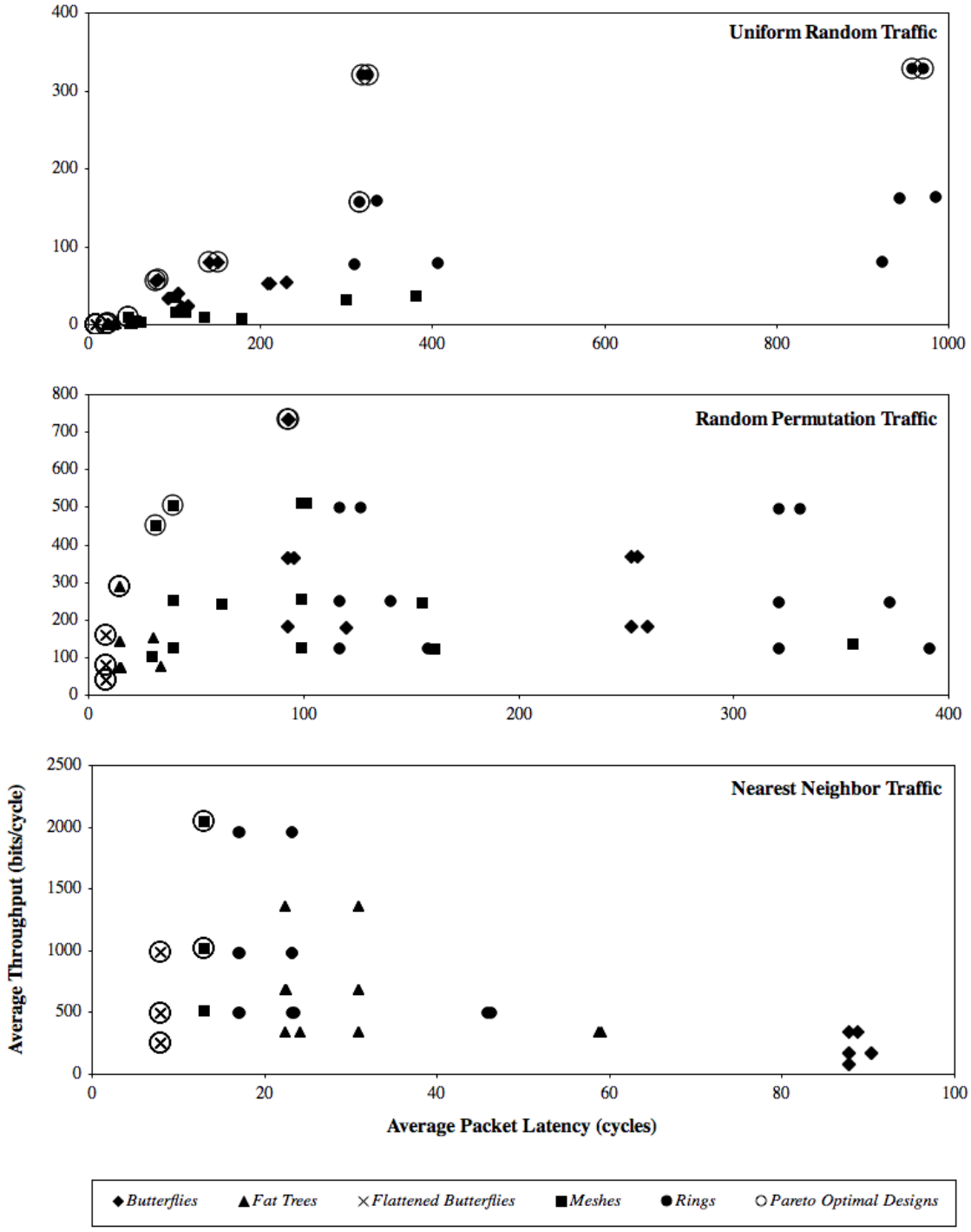

Figure 1. On-chip Interconnect Performance: Each of the three graphs above plots the average latency and throughput for each network in the design space defined by Table 1. The three graphs correspond to the three network traffic patterns. The circled network designs represent pareto optimal designs subject to an area budget of 32mm^2 (15% of a $15mm \times 15mm$ die). Those circled points that appear stacked actually have different latencies which the scale of these plots makes imperceptible. No network is always optimal, and thus, when selecting which network to implement in a multi-application environment, there must necessarily be a compromise amongst the different designs.

Network Design Space Parameters			
	Description	*Symbol*	*Range*
	Number Of Terminals	N	16,64,256,1024
	Topology	T	$\begin{cases} butterfly\ (k=2) \\ fat\ tree\ (k=2, levels=3) \\ flattened\ butterfly\ (k=2) \\ mesh \\ ring \end{cases}$
	Routing Algorithm	R	deterministic
	Arbitration Algorithm	A	store-and-forward, wormhole
	Message Size	M	256
	Packet Size	P	$32, 64, 128$
	Switch Packet Queue Capacity	SQ	$4, 16, 64$
	Converter Packet Queue Capacity	CPQ	$4, 16, 64$
	Converter Message Queue Capacity	CMQ	4
Synthesis-Analytical Area Model			
	Description	*Symbol*	*Value*
Synthesis	Queue area	$Queue_{area}$	$0.00002\ mm^2/bit$
	Wire pitch	χ	$0.00024mm$ [15]
	Crossbar area	$XBar_{area}$	$\chi^2 \times D_{in} \times D_{out} \times P^2$ [8]
	Wire overhead	W_{area}	20%
Analytical	Number Of Switches	S	$\begin{cases} \frac{N}{k} \times log(N) & \text{if } T \text{ is } butterfly \\ \frac{N}{k} & \text{if } T \text{ is } flattened\ butterfly \\ N + \frac{N}{k^2} + \frac{N}{k^4} & \text{if } T \text{ is } fat\ tree \\ N & \text{if } T \text{ is } mesh \\ N & \text{if } T \text{ is } ring \end{cases}$
	Switch Degree	D	$\begin{cases} k + log_k(\frac{N}{k}) & \text{if } T \text{ is } flattened\ butterfly \\ k & \text{if } T \text{ is } butterfly \\ \begin{cases} 2 & if\ leaf \\ k^2+1 & if\ internal \\ k^2+4 & if\ root \end{cases} & \text{if } T \text{ is } fat\ tree \\ 5 & \text{if } T \text{ is } mesh \\ 3 & \text{if } T \text{ is } ring \end{cases}$
	Switch Queue Area	SQ_{area}	$SQ \times P \times Queue_{area}$
	Switch Area	S_{area}	$SQ_{area} \times D + XBar_{area} + SQ_{area} \times D$
	Converter Message Queue Area	CMQ_{area}	$CMQ \times QPB$
	Converter Packet Queue Area	CPQ_{area}	$CPQ \times QPB$
	Converter Area	C_{area}	$2 \times CMQ_{area} + 2 \times CPQ_{area}$
	Network Component Area	N_{area}	$S \times S_{area} + N \times C_{area}$
	Total Network Area	I_{area}	$1.20 \times N_{area}$

Table 1. NoC Design Space and Area Model: The top part of this table defines a design space of on-chip network designs to explore. The space includes varying network topologies, arbitration algorithms, and resource provisions. Below, is the hybrid synthesis-analytical area model we use to estimate the circuit area of each network in the design space. We ground the area estimates in 90nm standard cell ASIC synthesis data, and then analytically combine these synthesized values to complete an estimate of total area.

tion, some communicating nodes are going to be at a distance in the network, and thus, the non-neighbor connections proffered by the higher levels of the tree speed that traversal. The differences in throughput amongst the fat tree designs on this workload are entirely attributable to packet size: the larger the packet the higher the throughput. This is feasible under the area budget, because the network does not require particularly deep queues on this workload. The same is true of the mesh network, for which the best designs incorporate the large, 128-bit packets and short, 4-entry queues to maximize throughput under the area budget. However, on average, the packet latency through the mesh is slightly higher due to the neighbor-only links in the mesh.

By contrast, the local adversarial traffic experiences exactly the opposite result. While random traffic latency *suffered* on neighbor-only topologies, the local adversarial traffic, which *is* neighbor-only, took good advantage of those topologies. Thus, for this workload, the mesh topologies are optimal, with the ring not far behind.

While there is a network to fit each workload, *there is no network to fit all workloads*. In other words, no network in the design space is optimal across all three workloads. The

optimal designs can differ in topology, and when the topologies are the same, the resource provisioning is very different. Although these workloads are synthetic, it would not be farfetched to encounter three similar patterns in a single multicore device, depending on the application or input data set that is running. Thus, in selecting a network, one will necessarily have to sacrifice performance on one or more applications.

4 Polymorphic On-chip Network Architecture and Design

The previous section empirically motivated the fact that no single fixed-design on-chip network efficiently communicates different styles of traffic. In this section we describe the polymorphic on-chip network, which can be configured at runtime to mimic traditional fixed function networks. From a hardware standpoint, the network is built from a sea of resources, namely buffers and crossbars. Careful design allows post-fabrication or even runtime configuration of these resources to form an interconnect with a custom topology, buffer allocation and packet size. We begin by describing the microarchitecture of the configurable fabric, followed by some examples of how to take advantage of the polymorphism.

4.1 Microarchitecture

The polymorphic network switch, as with a classic switch, consists of input and output packet queues, routers, arbiters and a crossbar of connections. We call the set of input queue, router, arbiter a *slice*. These slices are clustered into regions. Although Figure 2 illustrates eight slices per region, in the following section we will explore different configurations of this underlying fabric structure.

The crossbar connecting the slices in a region is a double crossbar that allows redefinition of the input connections as well as the usual dynamic definition of output connections. The diamond-shaped intersections on the incoming packet lines in Figure 2's slice detail denote the static connections, while the circular intersections indicate the dynamically switching output connections. Although Figure 2 shows these crossbars as bidirectional, they in fact consist of two single directional wires, because drive buffers can be employed only on directed wires. Furthermore, the physical crossbars in this crossbar are segmentable, with a potential segment point between each of the slices.

The configurable connection between input wires and crossbars, and the ability to segment a crossbar make it easier to construct switches out of slices from *different* regions in the fabric. This ability to aggregate slices from different regions into one logical switch turns out to be very valuable, as it allows for a dense packing of switches. This has two benefits: First, as the input and output queues consume most of the area of this fabric, it is important to waste as few as possible when configuring the network. Second, the closer the switches are physically, the shorter the routed links that connect them must be, leading to less link routing congestion and less capacitive load on those links.

Routing and arbitration: Each network slice contains routing and arbitration logic. These two pieces of logic serve the same function as the router and arbiter in their classic switch counterparts. The router is associated with an input queue, and determines to which output to route the first packet in the queue. Meanwhile each output queue has an associated arbiter which grants access to the output queue, one input at a time.

In the polymorphic network, we use source routing, where each packet carries its pre-computed route with it, rather than making routing decision at each point in the network. This allows the network to support any static routing scheme, such as the ones explored in design space exploration presented in Section 3. However, the effect of this is that it adds bits to the packet header, reducing the payload of a fixed-size packet. One way to get around this, and the one we currently utilize, is to implement wormhole arbitration [9], allowing a lead packet to carry the route and establish necessary connections, to be used by the subsequent packets belonging to the same message.

Configurable link resources: Surrounding the regions are configurable link tracks which support configurably routed links between the switches. Unlike the crossbar connections between the slices, the connections in these crossbars are statically configurable. They will implement one fixed topology per configuration.

Virtual channels: Each slice supports two virtual channels, which can share the available buffer space, using a flexible sharing scheme [7]. Networks requiring more than two channels, must aggregate multiple slices together.

Interface to cores: The polymorphic fabric will be connected to cores in much the same way as a normal network on chip would. A region is specialized further such that certain fixed input/output nodes are directed not to general purpose FIFOs, but instead to the target end-points.

Fabric configuration: We envision configuring the polymorphic fabric via bitstream. As opposed to FPGAs, which are programmed from external sources, when an application is loaded on a chip with this fabric, the OS, or runtime system, can reconfigure the network accordingly, by writing to memory-mapped bits containing the configuration as it would any other I/O device. The device could also include a default network configuration to support applications which do not specify a custom interconnect.

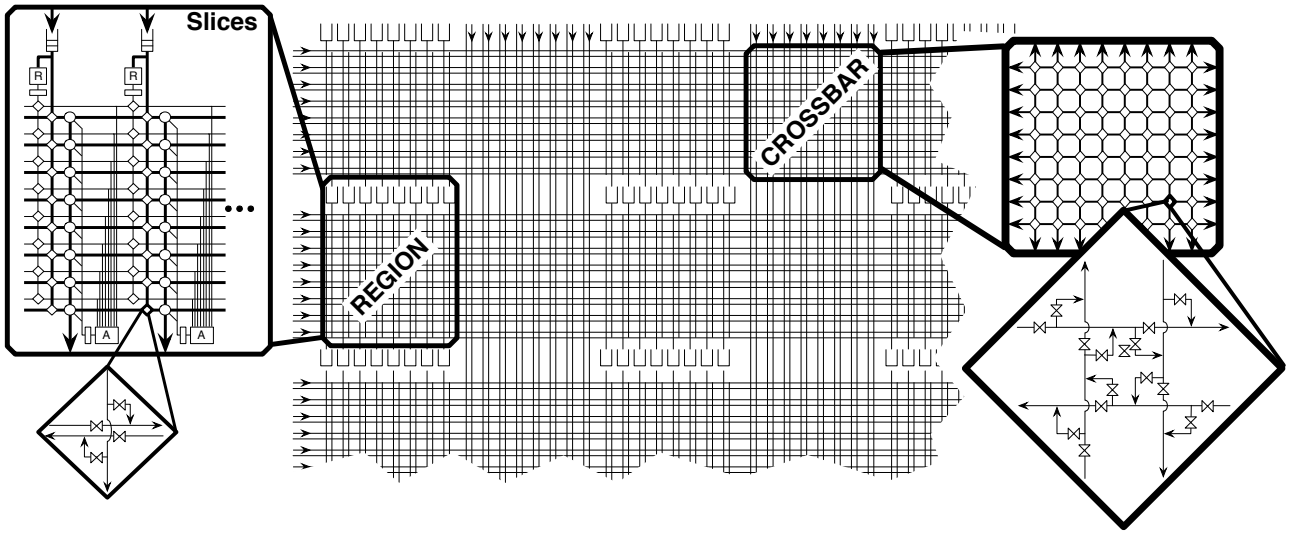

Figure 2. Polymorphic Network-on-Chip Microarchitecture: The polymorphic network is formed from a repeating pattern of "regions" and "crossbars". The connections within the crossbar are statically configurable, as shown in the detail above. The regions consist of multiple "slices", with each slice containing a slice of a network switch: an input queue, router, potential switch crossbar connections, and an arbiter.

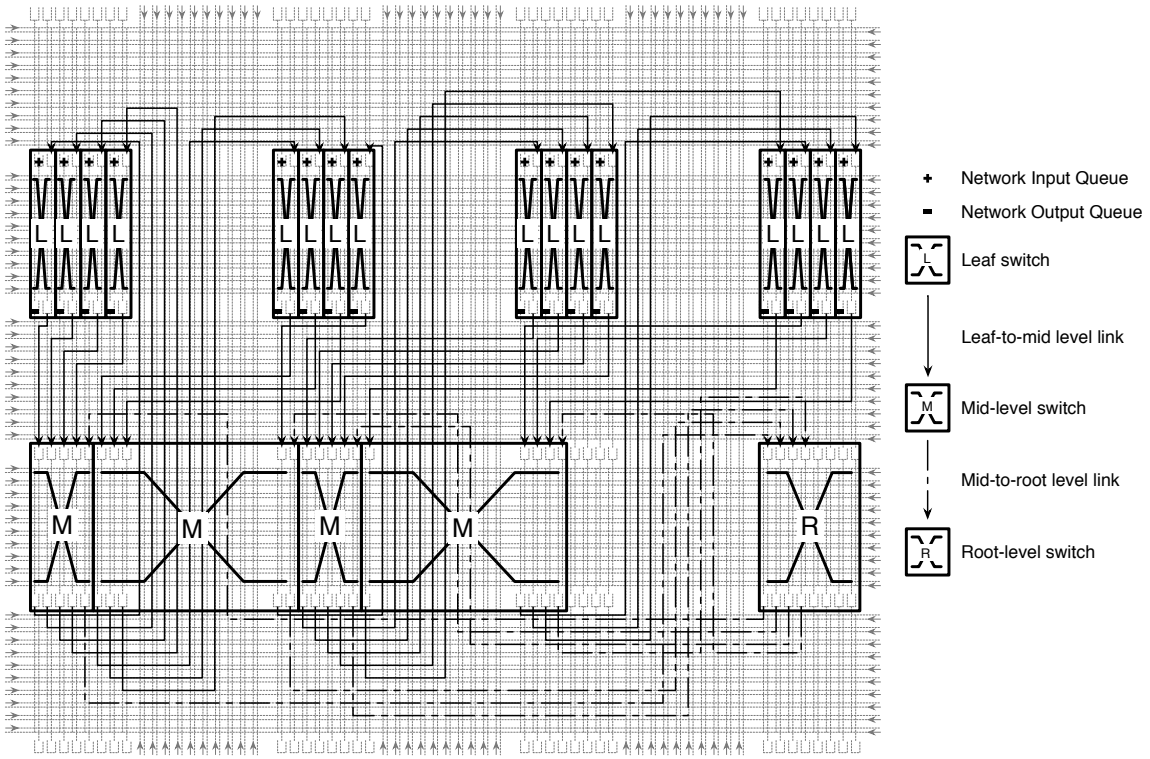

Figure 3. Fat tree topology on polymorphic fabric: The placement of switches and routing of links of a piece of a fat tree topolgy are shown on the polymorphic network fabric for a four-way fat tree with three levels. Only the subtree beneath beneath one root node is shown. Not shown are the links that form the mesh connection between root nodes. However, the available configurable links to the north, south and east of this region have sufficient leftover space to route those connections.

4.2 Examples of Polymorphism

The fabric described above introduces several modes of flexibility into the network, including flexible topologies, datapath widths, and queue depths, which we demonstrate here.

Flexible or hybrid topologies: Configuration of the fabric to implement a particular topology is a matter of forming appropriately sized switches and connecting them according to the desired topology. Figure 3 illustrates the mapping of a portion of a fat tree onto the fabric. It is a four-way fat tree with three levels: the same topology used in the design space exploration in Section 3.1. For legibility, we show the subtree of only one root node. In a full mapping, the leftover routing resources on the north, south and east edges of this diagram would be used to connect multiple nodes together. Note that in the regions occupied by this design, only six of the 128 queues have gone unused. This high utilization is enabled by the ability to form switches from slices assembled from multiple adjacent regions. We have demonstrated the instantiation of a fat tree, with highly-connected, high-degree switches, because it demands more careful configuration of the fabric.

Flexible buffer allocation: One can also increase the logical switch buffer size using this polymorphic fabric. It is possible to configure a single slice to connect the input queue to the output queue. Such a configuration forms a single logical buffer from two physically separate ones, in which packets advance from the back half to the front half automatically, as necessary. One could also view such a configuration as a 1×1 switch, which in fact does not route or switch, but simply advances packets forward as a queue would.

The decision to increase switch buffer resources need not be a universal decision. One can selectively increase buffer resources on certain switches or ports as the application may demand. This capability dovetails nicely with other research that develops tools to identify the ideal network configuration for a particular application [16, 23, 13].

Flexible packet sizing: In a similar manner, one can also increase the network packet size. Instead of aggregating sequential buffers, as in the previous example, one can aggregate parallel buffers and links. In effect, this aggregation increases the network's packet size and datapath width. Both of these aggregation techniques increase the per-switch, or per-link resource requirements of the fabric. However, with the polymorphic fabric, the increased resource demand is commensurate with the analogous increase in resource needs, in buffer sizes and wire bit widths, of a classic ASIC interconnect.

Note that the polymorphic network need not be restricted solely to general-purpose interconnect configurations. Instead it can support nearly arbitrary application-by-application tailoring. The performance benefits of such customization are well-documented in the literature [13, 12, 22], but were previously unattainable on multi-application devices.

4.3 Polymorphic Network Design Space

The previous section presented a general architecture for a polymorphic network, however it left all of the design parameters, such as packet widths and queue capacities unspecified. Figure 4 illustrates the architectural parameters which must be decided prior to implementing a polymorphic fabric.

There are N slices to a region, each containing a queue that is W bits wide and D entries deep. Each region has two crossbars that are $N \times H$ where H, the number of horizontal crossbars, must be at least N. Similarly, each configurable link track intersection is $V \times H$. As with the original network design space, we explore the parameter space of polymorphic fabrics. The first section of Table 2 articulates the polymorphic network parameter range we explore, totaling 108 polymorphic network instances.

Based on the parameters, we can calculate the area of the *base element* of the fabric. The base element consists of a region and a crossbar and is the minimal unit of fabric that can be replicated to form a larger swath of fabric. The second section of Table 2 details how the area of the queues and crossbars are calculated and totaled to compute the area of a particular instance of the polymorphic fabric. In a region there are N queues, and beneath them, two crossbars. One crossbar from the input lines to the horizontal crossbars which is $N \times H$, and one crossbar from the horizontal lines to the output queues which is $H \times N$. Between regions is a second crossbar. It consists of four crossbars, two of size $H \times 2V$ for packets entering in the horizontal direction, from the left or right, and two of size $V \times 2H$ for packets entering in a vertical direction, from above or below.

The number of base elements needed to implement a particular network, as well as the resulting latency and throughput of the network, depend on how the polymorphic fabric is configured. This is where the last parameter, the network configuration C, comes in. To determine how much polymorphic fabric a particular network configuration will require, we calculate how many queues each of its switches will require. This calculation was performed manually for 2×2 switches up to 8×8 switches in each polymorphic fabric. These mappings accounted for the spatial blowup that occurred when the switch required more bisectional bandwidth than was available horizontally in a single region. These initial queue counts were increased proportionally if the network configuration demanded deeper or wider queues than the queues provided by the polymorphic fabric. Note that in the reverse case, because queues in the polymorphic fabric cannot be subdivided, when the network configuration's queues are smaller than those in the fabric, the extra

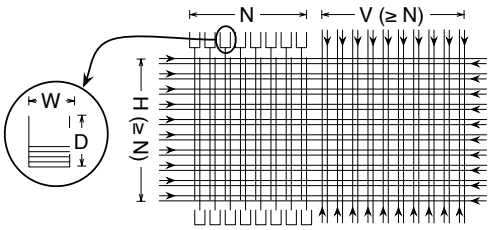

Figure 4. Polymorphic Fabric Design Parameters

Polymorphic Network Fabric Design Space Parameters			
	Description	*Symbol*	*Range*
	Number of slices per region	N	2,4,8,16
	Queue width	W	32,64,128
	Queue depth	D	4,16,64
	Width of horizontal configurable link tracks	H	$N, 2 \times N$
	Width of vertical configurable link tracks	V	N,$2 \times N$
	Network configuration	C	each network in Section 3.1
Polymorphic Fabric Configuration Model			
	Description	*Symbol*	*Value*
	Base number of polymorphic queues	$PQueues_{base}$	manual switch configuration
	Queue depth adjustment	$Adjustment_{depth}$	$\begin{cases} \frac{C_{QueueDepth}}{D} & \text{if } C_{QueueDepth} > D \\ 1 & \text{otherwise} \end{cases}$
	Queue width adjustment	$Adjustment_{width}$	$\begin{cases} \frac{C_{QueueWidth}}{W} & \text{if } C_{QueueWidth} > W \\ 1 & \text{otherwise} \end{cases}$
	Adjusted number of polymorphic queues	$PQueues_{adjusted}$	$PQueues_{base} \times Adjustment_{depth} \times Adjustment_{width}$
	Number of base elements needed	$BaseElements_{needed}$	$PQueues_{adjusted}/N$
Polymorphic Fabric Area Model			
	Description	*Symbol*	*Value*
Synthesis	Queue area	$Queue_{area}$	$0.00002\ mm^2/bit$
	Wire pitch	χ	$0.00024mm$ [15]
	Crossbar area	$XBar_{area}(D_{in}, D_{out})$	$\chi^2 \times D_{in} \times D_{out} \times W$
Analytical	Region area	$PolyRegion_{area}$	$N \times W \times D \times Queue_{area} + XBar_{area}(N, H) + XBar_{area}(H, N)$
	Crossbar area	$PolyXBar_{area}$	$2 \times XBar_{area}(H, 2 \times V) + 2 \times XBar_{area}(V, 2 \times H)$
	Total area of base element	$BaseElement_{area}$	$PolyRegion_{area} + PolyXBar_{area}$
	Area of polymorphic fabric to implement C	$Fabric_{area}$	$BaseElements_{needed} \times BaseElement_{area}$

Table 2. Polymorphic Fabric Design Space, Configuration Model, and Area Model: The "Polymorphic Network Fabric Design Space Parameters" section of this table enumerates the parameters that define the polymorphic fabric design space, with the role of the parameters illustrated in Figure 4. The "Polymorphic Fabric Configuration Model" indicates how many base elements of the polymorphic fabric are required to instantiate a particular configuration C. Finally, the model translating this base element count to area is outlined in the last, "Polymorphic Fabric Area Model", section.

queue capacity amounts to wasted area. Finally, for each network topology, we account for any extra polymorphic fabric required to implement the overall routing.

4.4 Polymorphic Network Design Space

We explore the design space by varying the parameters outlined in the section of Table 2 labelled "Polymorphic Network Fabric Design Space Parameters". In each fabric, we instantiate each of the networks from the initial network design space (from Section 3), counting how many base elements are required to implement each network according to the model in the section of Table 2 labelled "Polymorphic Fabric Configuration Model". Finally, we translate these base element counts to area with the "Polymorphic Fabric Area Model" from Table 2. Figure 5 plots the average area expansion of each possible polymorphic fabric design across all of the network configurations.

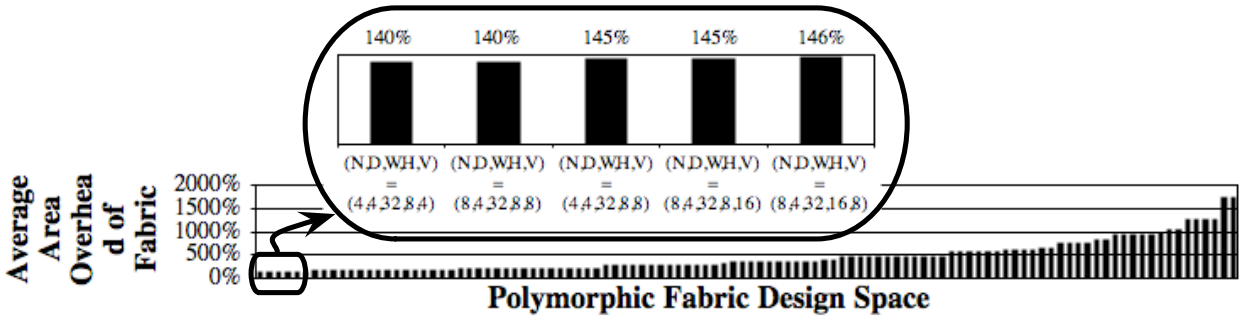

Figure 5. Polymorphic Network Design Space: We compare the area efficiency of the polymorphic network design points. This chart shows the wide range of area overheads, from 38% on up. The fabric designs that incur the largest overhead are those that overprovision the polymorphic queues, in both depth and breadth. While fabrics with small queues can easily mimic large queues, fabrics with large queues waste a great deal of area when only small queuse are needed. The next most significant overhead determinant is horizontal routing resources. When severly constrained, this incurs a large increase in the required polymorphic resources.

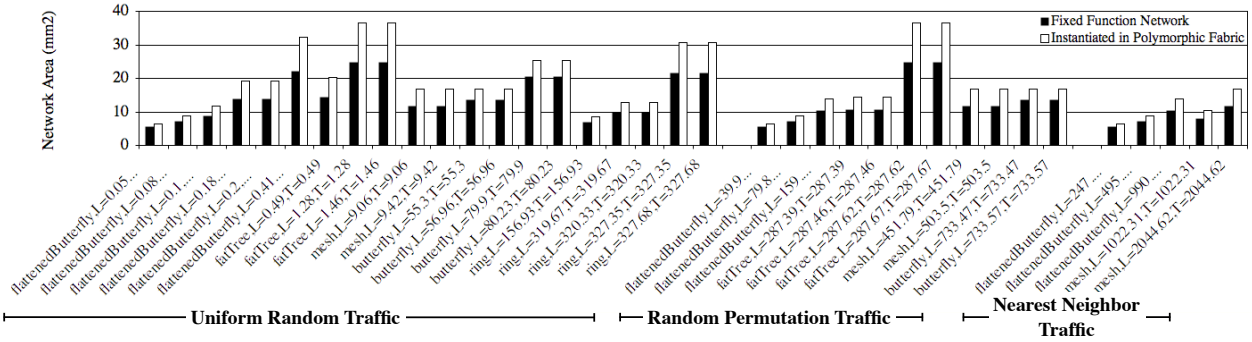

Figure 6. Polymorphic Overhead Across Networks: The black bars in this chart indicate the area of each of the pareto optimal fixed-function networks from Section 3.2. The white bars next to them indicate the area required by a polymorphic network to implement each network in a polymorhpic network. A polymorphic network of a given size can implement any network design whose white bar is within the area budget.

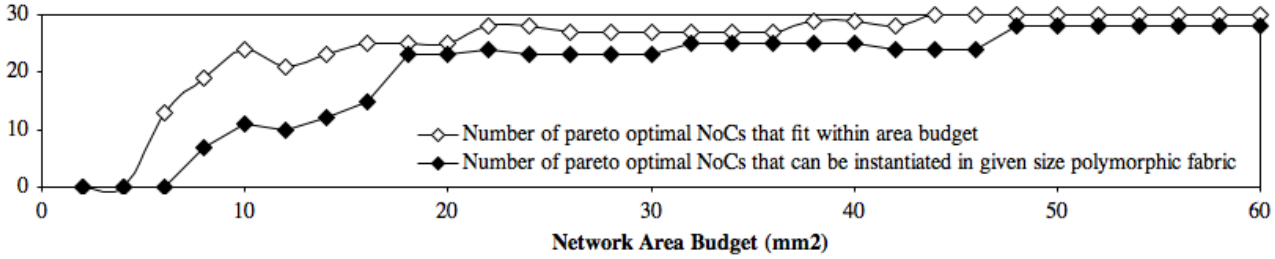

Figure 7. Polymorphic Network Coverage Sensitivity: We examine how sensitive the polymorphic network design coverage is to the area budget. When the area budget is low, a relatively small percentage of the feasible fixed-function interconnects could also be implemented in a polymorphic fabric. Thus, a polymorphic fabric is likely not the best interconnection option in this case. However with a more standard area budget, of 5-10% of die area, the polymorphic fabric coverage improves to support implementation of 80% of all fixed-function interconnect options. This coverage improves even further, to 97%, with an increasing area budget.

Each black bar in Figure 5 corresponds to one polymorphic fabric design. The height of the bar represents the area of the polymorphic fabric relative to the ASIC area of the network it is implementing. There are several overriding trends which govern the quality of a polymorphic fabric.

- The "worst" fabrics, those that incur the greatest increases in area consumption, are the fabrics which overprovision the polymorphic fabric queues, both in depth and width, relative to the queues in the configuring network.
- For a fixed queue size, the more constrained the horizontal routing resources, the greater the area overhead.
- The number of slices per region and the amount of vertical routing do not influence area overhead with any notable pattern. The reason vertical routing resources are less significant than horizontal routing resources, is because horizontal routing resources are used to build network switches and route horizontally, while vertical routes are used only for routing.

We highlight the "best" fabrics on the left hand side of Figure 5. These instances of the polymorphic fabric incur, on average, the smallest area overhead. In keeping with the overall trends, the best polymorphic designs implement narrow and shallow queues, with generous routing resources in the horizontal and vertical directions. This minimizes waste because queues are aggregated only when the instantiated network design requires it. The most area-efficient polymorphic fabrics incur a 40% area overhead compared to the average of the ASIC network designs.

5 Polymorphic Network Evaluation

5.1 Area Overhead on Optimal Networks

We identified the most efficient polymorphic fabric, one with four slices per region, four, 32-bit packets per queue, eight horizontal and four vertical routing tracks ($N = 4, D = 4, W = 32, H = 8, V = 4$). We then examine the amount of this polymorphic fabric required to implement each pareto optimal network design. Figure 6 plots these results. Each pair of bars corresponds to a network design, the height of the black bars indicating its area when fabricated directly, and the height of the white indicating the amount of polymorphic fabric required to implement the same network. This illustrates the flexibility of the polymorphic fabric. Once you have committed to spending a certain amount of area on a polymorphic fabric (corresponding to some point on the Y axis of Figure 6), you can implement *any* network for which the corresponding white bar does not exceed that area budget. This chart includes the pareto optimal designs from our initial design space exploration, however the network configurations need not be constrained to be one of them.

Figure 7 compares how many pareto optimal networks can be implemented as directly in a fixed amount of silicon area versus how many can be instantiated in a polymorphic fabric of that size. This data indicates that when area is very limited, you are better off implementing a fixed-function network, because the flexibility of a polymorphic network is compromised. However, once approximately $18mm^2$ is allocated to the interconnect, a polymorphic fabric of a given size can implement the overwhelming majority of the viable (with respect to size) optimal fixed-function networks.

This data demonstrates that one should *always* build a polymorphic network, except in two cases: (1) when the application is fixed and well known ahead of time (e.g. embedded fixed-function devices); and (2) a tighter area constraint than $18mm^2$ is required. Of the fixed topology networks we studied, no configuration (queue size, packet size, etc) exists that is under our chip wide 15% area budget, and achieves a performance superior to a polymorphic network. For any pareto-optimal fixed configuration network with area less than 15% of chip area, we can instantiate a polymorphic network that is also less than 15% of chip area that achieves the same performance. One can exploit the adaptability of the polymorphic network and reconfigure it at runtime to implement the network that best matches an application's demands, thereby improving total system performance.

6 Conclusion

This paper has identified the need for a flexible on-chip interconnect architecture, and thus proposed a polymorphic network architecture. This network can be configured on a per-application basis to function as the network that best suits the application needs, thereby improving performance across a range of applications as compared to a fixed function network.

We have explored the architectural design space of the proposed polymorphic network, identifying the design point which provides the most efficient network configurations. While polymorphic network design incurs an average of 40% area overhead relative to a fixed function network, it provides significant flexibility. Withing a fixed area budget spent on a polymorphic fabric, one can implement the vast majority of all fixed-function networks that would have fit within the area budget.

The value of application-specific interconnect tailoring has already been demonstrated. As programmable multicore devices emerge, a polymorphic network – enabling tailoring of network configurations to fit application traffic patterns – stands to offer a valuable boost to overall device performance.

Acknowledgements

This work has been made possible through the generous support of the Gigascale Systems Research Center, an NSF CAREER Award (ACR-0133188), Sloan Research Foundation Award (Oskin), Intel Fellowship, Google Anita Borg Scholarship (Kim), and support from Intel and Dell. We would like to thank our reviewers for their helpful feedback.

References

[1] P. Abad, V. Puente, J. A. Gregorio, and P. Prieto. Rotary router: an efficient architecture for CMP interconnection networks. *ACM SIGARCH Computer Architecture News*, 35(2):116–125, 2007.

[2] A. Abnous, H. Zhang, M. Wan, G. Varghese, V. Prabhu, and J. Rabaey. The pleiades architecture. *The Application of Programmable DSPs in Mobile Communications*, pages 327–360, 2002.

[3] J. Balfour and W. J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *Proc. of the International Conference on Supercomputing*, pages 187–198, 2006.

[4] K. Constantinides, S. Plaza, B. Z. Jason Blome, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky. Bulletproof: A defecttolerant CMP switch architecture. In *Proc. of the International Symposium on High-Performance Computer Architecture*, pages 5–16, 2006.

[5] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra. Spidergon: a novel on-chip communication network. In *Proc. of the International Symposium on System-on-Chip*, page 15, 2004.

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

[7] W. J. Dally. Virtual-channel flow control. In *Proc. of the International Symposium on Computer Architecture*, pages 60–68, 1990.

[8] W. J. Dally and J. W. Poulton. *Digital systems engineering*. Cambridge University Press, New York, NY, USA, 1998.

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

[10] J. D. Davis, J. Laudon, and K. Olukotun. Maximizing CMP throughput with mediocre cores. In *Proc. of the International Conference on Parallel Architectures and Compilation Techniques*, pages 51–62, 2005.

[11] C. Ebeling, D. C. Cronquist, and P. Franklin. RaPiD - reconfigurable pipelined datapath. In *Booktitle of the International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, pages 126–135, 1996.

[12] C. Grecu and M. Jones. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Transactions on Computers*, 54(8):1025–1040, 2005.

[13] J. Hu, U. Y. Ogras, and R. Marculescu. System-level buffer allocation for application-specific networks-on-chip router design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 25(12):2919–2933, December 2006.

[14] J. Huh, D. Burger, and S. W. Keckler. Exploring the design space of future CMPs. In *Proc. of the International Conference on Parallel Architectures and Compilation Techniques*, pages 199–210, 2001.

[15] International technology roadmap for semiconductors. Semiconductor Industry Association, http://public.itrs.net/, 2006.

[16] A. Jalabert, S. Murali, L. Benini, and G. D. Micheli. xpipesCompiler: A tool for instantiating application specific networks on chip. In *Proc. of the Conference on Design, Automation and Test in Europe*, page 20884, 2004.

[17] J. Kim, W. J. Dally, and D. Abts. Flattened butterfly topology for on-chip networks. In *Proc. of the International Symposium on Microarchitecture*, pages 172–182, 2007.

[18] N. Kirman, M. Kirman, R. K. Dokania, J. F. Martinez, A. B. Apsel, M. A. Watkins, and D. H. Albonesi. Leveraging optical technology in future bus-based chip multiprocessors. In *Proc. of the International Symposium on Microarchitecture*, pages 492–503, 2006.

[19] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha. Express virtual channels: towards the ideal interconnection fabric. In *Proc. of the International Symposium on Computer Architecture*, pages 150–161, 2007.

[20] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, D. Hillis, B. C. Kuszmaul, M. A. S. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak. The network architecture of the connection machine CM-5 (extended abstract). In *Proc. of the Symposium on Parallel Algorithms and Architectures*, pages 272–285, 1992.

[21] R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. In *Proc. of the International Symposium on Computer Architecture*, page 188, 2004.

[22] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. D. Micheli, and L. Raffo. Designing application-specific networks on chips with floorplan information. In *Proc. of the International Conference on Computer-Aided Design*, pages 355–362, 2006.

[23] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. D. Micheli, and L. Raffo. Designing application-specific networks on chips with floorplan information. In *Proc. of the International Conference on Computer-Aided Design*, pages 355–362, 2006.

[24] H. Singh, G. Lu, E. Filho, R. Maestre, M.-H. Lee, F. Kurdahi, and N. Bagherzadeh. MorphoSys: case study of a reconfigurable computing system targeting multimedia applications. In *Proc. of the Conference on Design Automation*, pages 573–578, 2000.

[25] H. Wang, L.-S. Peh, and S. Malik. Power-driven design of router microarchitectures in on-chip networks. In *Proc. of the International Symposium on Microarchitecture*, pages 105–115, 2003.

[26] A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, and U. Catalyurek. A scalable distributed parallel breadth-first search algorithm on bluegene/l. In *Proc. of the Conference on Supercomputing*, page 25, 2005.