High Performance Networks for Dataflow Architectures

Pravin Bhat Andrew Putnam
{pro, aputnam}@cs.washington.edu
University of Washington
Seattle, WA

Abstract

Dataflow microprocessors have emerged as a viable solution to the near-term demise of traditional superscalar processor architectures. The performance of these dataflow architectures is heavily dependent on the network design, hence good network performance is paramount. The network must also provide fault tolerant, deadlock-free operation while allowing scaling from one to several hundred nodes. The unique demands of dataflow processors on network resources, as well as an increasingly complex physical environment, make it difficult to determine the optimum network architecture.

Our research evaluates the network design for the WaveScalar dataflow architecture [1], though the same concepts extend to other dataflow architectures such as Monarch [17]. We evaluate the effects of different topologies along with various routing and flow control strategies to find the best performance within the available chip area constraints. We find the optimal design is a torus topology with adaptive routing using a link bandwidth of 2, 2 virtual channels, and 2-4 entries per queue. Our design corrects deficiencies in the current WaveScalar design while adding negligible area overhead and doubling performance on certain workloads.

1. Introduction

Current technological trends suggest that within 5-10 years conventional microarchitectures will be impractically complex and only marginally faster than current processors [5]. Chip Multiprocessors (CMPs) have been proposed as a way to reduce complexity and improve performance in future architectures [1,6,7]. CMPs require an on-chip network for core-to-core communication. One particular kind of CMP called a dataflow processor [1, 17] is a particularly attractive solution to these problems, yet these architectures have many characteristics that complicate their design and implementation

In this paper, we present an on-chip network design for dataflow architectures that addresses these issues, and satisfies the design constraints of high performance per unit area, fault tolerance, guaranteed message delivery, and deadlock-free operation.

On-chip networks must provide much of the same functionality as traditional multi-computer networks, but must do so under very different constraints. Along with different constraints come a different set of metrics that are used to evaluate the success of the network design. On top of the traditional metrics of speed, reliability, scalability, and cost-effectiveness, on-chip networks must efficiently use die area. Minimizing die area limits the number of links, the bandwidth, and the buffer sizes that can be used at each networked component. This means that the viability of a network is always related to its area, so traditional viability metrics like speed become speed per unit area.

These networks require similar functionality as is provided by an Ethernet connection running TCP/IP, yet have unique performance, complexity, and timing constraints that prohibit protocols with large timing and buffering overhead.

We investigate network designs for the dataflow architectures similar to WaveScalar [1], a processor that is being developed by the computer architecture group at the University of Washington. Unlike most other CMP designs that have tens of relatively complex processor cores, WaveScalar consists of hundreds to thousands of very simple processor cores. With such a large number of cores, fast, efficient communication between the processor cores is essential. WaveScalar is also a dataflow processor, which makes it difficult to predict when and where network resources will be needed. On-chip networks like the one we are designing for WaveScalar have been proposed, and some of the concepts behind those designs are certainly applicable to the WaveScalar network [4]. However, most on-chip network research focuses on manufacturing processes much larger than those that will be used for WaveScalar. There is significant reason to believe that the on-chip network designs that worked in the larger process sizes will not be practical for deep sub-micron processes just like conventional microarchitectures do not work in deep sub-micron processes. Also, manufacturing defects increase exponentially in smaller process sizes, so the networks must be adaptable to manufacturing defects in order to allow commercially viable chip yields. The most challenging portion of the WaveScalar network design is providing deadlock-free operation in a network that does not support dropping messages. As a dataflow processor, messages have an ordering relationship that causes a number of potential deadlock situations. While virtual channels provide a mechanism for routing messages around congested routes, there are cases when the receiving processor can only continue if it receives a particular message. The virtual channels must be set up so that these messages can bypass other messages in order to guarantee forward progress

1.1. Dataflow Networks

Dataflow architectures add additional constraints to on-chip networks that complicate the network design. Most importantly, dataflow architectures display unpredictable traffic patterns, exhibit highly bursty traffic, and do not guarantee message consumption upon message arrival.

1.1.1 *Unpredictable Traffic*

Traditional Von Neumann architectures that have a program counter and structured pipeline exhibit predictably regular operation that allows network resources to be allocated before the resources are needed. This results in resource arbitration that is off the critical path, and hence does not limit performance of the processor.

Unlike Von Neumann architectures, dataflow architectures operate on the dataflow firing rule: execute instructions as soon as all of the inputs to that instruction are available. There is no program counter or pipeline that regulates the demand for network resources. Consequently, network resources must be allocated at the time they are needed. This allocation can become a bottleneck, and hence must be done as quickly as possible. We propose a mechanism for dealing with resource allocation in section 2.3

1.1.2 *Bursty Traffic*

Dataflow programs expose massive amounts of parallelism to the architecture. Any dataflow program can be represented using a dataflow graph. This dataflow graph can be broken into a set of independent dataflow subgraphs that can be executed in parallel.

We have observed that there numerous dataflow subgraphs that are data dependent upon the same input. When this input is produced at the end of some other subgraph, the result is forwarded to all of the dependent subgraphs. Ideally, the input can be forwarded to all of the dependent subgraphs simultaneously, minimizing idle time. Given limited network resources, this may not always be possible. The overall performance of dataflow programs is dependent on minimizing unnecessary idle time, handling these heavy bursts is essential for facilitating fast execution.

1.1.3 *Unconsumed Messages*

Numerous routing algorithms have been proposed for on-chip networks that all claim to be deadlock free [3, 8, 10, 14]. The deadlock-free characteristic of these algorithms relies on the idea that a network node will consume any message that makes it to the node. This is not necessarily the case in a dataflow processor. Each processing element (PE) in a dataflow processor has a finite amount of buffer space with which to store operands while awaiting its partner operand. If these buffers fill up, then the PE cannot accept the network message without generating a network message of its own. This means that, without clever design, the network is prone to deadlock even if it uses a provably deadlock free routing algorithm. We propose additions to the network architecture that allow deadlock-free and livelock-free operation in 2.2.

1.2 Requirements of Dataflow Networks

Based on the characteristics of dataflow architectures from the previous section, and on technological concerns addressed in [19], the following are the minimum requirements for an on-chip network of dataflow processors:

1. High performance, especially in the presence of bursts
2. Area efficient
3. Guaranteed message delivery
4. Deadlock and livelock free
5. Fault tolerant

1.2.1 *Fault Tolerance*

The fault tolerance requirement warrants further discussion. Fault tolerance is a growing concern for chip designers [19]. As process technology continues to shrink, a number of factors arise that change the way in which designers must think about on-chip systems and networks. Recent projections for 2010 [18] predict:

1. Manufacturers will no longer be able to conduct in-factory testing and defect correction
2. On average, 20% of the transistors on each chip will be defective after manufacturing
3. An additional 10% of transistors will fail within the first few months of chip operation
4. Chips will continue to degrade over their lifetimes

Even if these projections turn out to be pessimistic, it is clear that fault-tolerant architectures will replace traditional architectures by the end of this decade. From this, we adopt the following design requirements for fault tolerance in dataflow networks:

1. There must be no single point of failure
2. The network must be able to adapt to faulty links by finding alternative routes

While describing and implementing a fully fault-tolerant network architecture is beyond the scope of this paper, we do suggest topologies and routing algorithms that are a first step toward a fault-tolerant network design.

1.3. *Organization*

The paper is organized as follows. Section 2 describes the selections of topology, routing, and flow control that are used for this study. Section 3 describes the performance sensitivity of varying certain parameters of the network architecture. Section 4 introduces a hardware implementation of the network, which allows for performance and area studies. Section 5 concludes the study.

2. Topology, Routing, and Flow Control

In this section, we evaluate different design choices for topology, routing, and flow control, and make selections as to which are feasible candidates for further evaluation.

2.1. Topology

In this section, we evaluate several different network topologies for physical viability. Only those topologies that are deemed viable will be studied in detail.

2.1.1. Physical Constraints

The physical layout of dataflow processors puts constraints on network topologies that make sense for multicomputer networks. This allows us to exclude certain network topologies from further consideration.

Dataflow processors, like most CMPs, consist of a collection of clusters laid out in a rectangular configuration with *R* clusters per row and *C* clusters per column for a total of *R*x*C* clusters. In most configurations, *R*=*C*, but this is not a requirement.

2.1.2. The Fully Connected Network

The fully connected network topology is the simplest topology and allows direct connection between all nodes. Unfortunately, this topology does not scale well for the number of nodes in the network. For *n* nodes, a bus width *w*, and a total number of connections (bandwidth) *b*, the fully connected topology requires $(n+1)*w*b$ wires into each network switch. For wires in the high metal layers, typical design rules [9] call for deep sub-micron processes call for wire widths of 5λ and wire spacing of 5λ. This means each wire will occupy 10λ.

For 130nm, the nearest-term technology node in which a dataflow processor may be manufactured, λ≈0.75μm. This leads to bus widths of slightly more than 2 mm above each cluster for the 4x4 cluster, single-message per cycle configuration (*n*=16, *w*=160, *b*=1). This means that the chip size is a minimum of 8mm on each side due to network wires alone. For the desired configuration of at least 2 messages per cycle (*b*=2), this means the chip is a minimum of 1.6 cm on a side. While this is not technically prohibitive, it is likely commercially prohibitive. Worse, this topology does not scale well to a larger number of clusters, which violates the scalability design goal of WaveScalar. For an 8x8 configuration, the chip would be approximately 62.4 cm on each side, which is obviously prohibitive. For these reasons, we exclude the fully connected topology from further analysis.

2.1.3. Tree Topology

The tree topology is a tempting network topology since the internal node network structure is generally a tree, and this would logically extend the same hierarchical relationships. The tree groups clusters into small groups, which reduces the number of nodes in the top-level network connection graph. This could allow small groups of fully connected clusters for fast inter-cluster communication, but without the scaling problems of the fully connected network topology. Depending on instruction placement, this topology may offer substantial benefit to programs that can run within one of these grouped clusters.

Unfortunately, the tree topology results in a single point of failure at the top node. Any failure in that network node will result in the chip being partitioned, which results in a dead chip. Due to the high likelihood of failure in future technologies, the tree topology cannot satisfy the fault tolerance requirement and thus will not be considered further.

2.1.4. Hypercube Topology

The hypercube has been extensively studied for its property of having a maximum distance between any two nodes in an *N*-node network of $\log_2(N)$. Studies have shown how to efficiently lay out hypercube networks in VLSI [12, 16], and that incomplete hypercubes retain the same qualities as complete hypercubes [20]. Still, the hypercube topology suffers from several downfalls that make it a poor choice for a dataflow network. First, incomplete hypercubes have nodes that can be cut off from all other nodes by a single faulty link. This violates the fault-tolerance criteria. Even if the dataflow processor only used complete hypercubes, they are difficult to scale to a large number of nodes. Most evaluations of efficient hypercube layouts deal with configurations of no more than 4x4. Future dataflow configurations could grow to several hundred nodes on each side, which is clearly too complex for any of the hypercube layout strategies currently proposed.

2.1.5. The Grid Topology

The grid topology is a simple topology that has the convenient attribute of uniform physical delay between hops since the length of wires between each network node is the same. This topology is also a natural extension of the physical wire routing available on chip. Each network node communicates with a maximum of 4 other nodes, one in each cardinal direction. Nodes at the edges of the grid connect to the L2 cache.

This design scales to any number of clusters, satisfying the scalability design requirement. The grid topology also provides a good mechanism for routing around defective nodes (as long as the routing is adaptive), which helps to satisfy the fault-tolerance design requirement. The grid topology has a number of favorable characteristics and no obvious fatal drawbacks, so it will be considered in this study.

2.1.6. Grid-like Topologies

There are a number of other potential topologies that are similar to the grid topology, but with additional connections between particular nodes. These topologies include, k-ary n-fly [15], torus, and twisted torus [22]. Each of these has non-uniform link length, which equates to non-uniform message passing delay in the physical domain. Also, long wires have super-linearly increasing wire delay as devices scale to smaller process sizes, so the gap between

communication over the short and long links will continue to grow. This negatively affects scalability. However, some of these topologies allow interesting routing characteristics that counteract this delay, and hence may still be viable. For this study, we consider only the torus topology and leave the other topologies for future work.

The torus topology has non-uniform latency, but has a bound on the longest link length. Communication speeds must accommodate the longest link, and hence communication in general is slowed. However, there are a number of routes that can be reached in a smaller number of hops, and hence the slower communication speed may be overcome by better route choices.

2.2. Deadlock-Free Operation

Traditional IP-based networks deal with the problem of deadlock by requiring the sender to buffer all network packets until the sender receives acknowledgement of receipt of the packet by the receiver. The sender uses a timeout value to decide when to retransmit a packet that has not been acknowledged. The network is thus free to drop packets when necessary. On-chip networks have one critical difference from traditional IP networks: extremely limited buffer space. If on-chip networks used IP, there could be very few instructions in flight at any given time. As a consequence, on-chip networks cannot drop messages. Without the ability for the network to drop messages, there is the possibility of deadlock.

A large number of provably deadlock-free network topologies and routing algorithms have been proposed for on-chip networks [8, 14, 15]. These algorithms overcome the buffer space limitation by making the assumption that receiving nodes will always accept messages. All of the deadlock-free proofs hinge on the assumption that a message is consumed as soon as it makes it to the destination node. This is unfortunately not the case in dataflow networks.

To illustrate how deadlock can occur in dataflow networks, we use the simple illustration of two network nodes as shown in Figure 1. While this is a simplified version of the full network, the deadlock problem extends to the full design. In this example, Node A wishes to send message M1 to Node B. However, Node A has stalled execution because the output queue is full from messages M2 and M3. The output queue will not drain because the messages are bound for Node B, which cannot execute and drain its input queue until message M1 is sent.

To get around this issue, there needs to be a way to flush the inputs out of Node B so that other inputs have the chance to get in and break the deadlock. In order to do this, there must be a way to selectively evict a message from the input queue and send it somewhere where it will eventually make it back to the node.

We present a design for overcoming deadlock in dataflow networks by using dedicated channels to and from memory. For correctness, these channels must be reserved strictly for memory traffic. Since memory is essentially infinite, the standard on-chip assumption that the destination node can accept all messages is true when sending messages to memory. This gives the network the guarantee that eventually, the blocking message will be evicted and the deadlock can be broken.

The design requires some additional logic to avoid the possibility of livelock. Livelock occurs in dataflow processors when messages are moving through the network, but the operand pairs that need to unite in order to execute never end up in the processing unit simultaneously. To avoid this case, we adopt an approach similar to that in the Manchester Machine [13]. We bias the input queue ejection mechanism to prefer ejecting the right-hand operand from the dataflow graph. This provides an increased rate of messages through the network for right-hand operands than left-hand operands. The two different rates guarantee that matching operands will eventually meet in the processing unit.

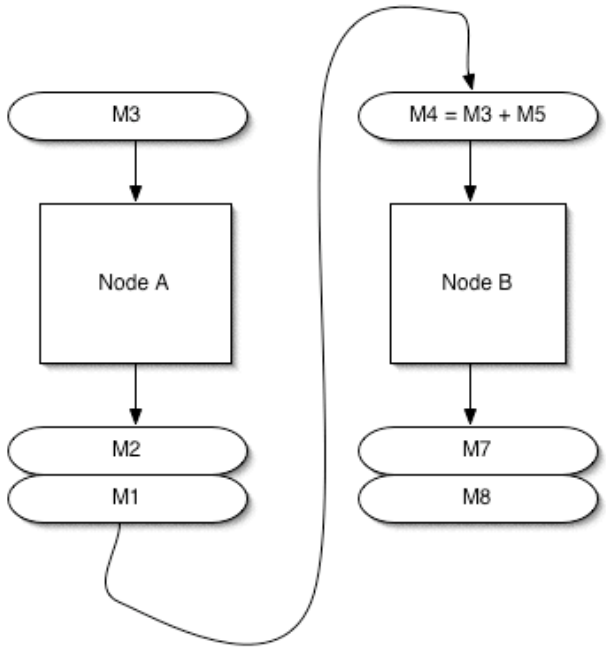

Figure 1 - Dataflow Deadlock: This interconnection network is blocked on message M3 which cannot get onto the network because Node A's outgoing queue is full, and the output instructions have nowhere to go.

While the deadlock breaking mechanism guarantees a way for the network to eventually break deadlock, it presents a new set of problems and drawbacks. First, dedicating two channels for memory traffic alone results in an 8-11% increase in area of each network switch. The

increased area does not result in improved performance, since the bandwidth of memory messages is rarely the bottleneck in dataflow programs [1]. We leave quantitative evaluation of this mechanism and real-world performance studies as future work.

Deadlock and livelock situations may seem far-fetched, but the situation can arise frequently in a large dataflow network. This frequency can be alleviated by using larger input caches, providing additional network bandwidth, and more intelligent output message handling schemes, but the vulnerability still exists. The handling mechanism does not have to be fast since the situation should be very rare, but it should be fast enough that it does not require an excessive amount of time to resume typical operation.

2.3. Flow Control

Flow Control is particularly important in dataflow processors due to the unpredictability of dataflow traffic. Without even a single cycle to do resource reservation, nodes in the network cannot arbitrate for resources and guarantee message reception. The round-trip latency required for handshake protocols would add prohibitive overhead on message transfers, so a different solution is necessary. In this case, the best mechanism is the stop channel mechanism introduced in [21].

The stop channel mechanism is chosen for its low buffering overhead requirements and its high message throughput in the absence of resource conflicts. In our system, stop channel flow control works as follows. First, the source sends a message to the receiver node, and simultaneously stores the message in a small buffer. The source node can continue to send messages as long as it saves the messages in case of rejection. The messages in the buffer must remain until an acknowledgement or rejection signal from the receiver. In our network, the receivers and senders are always only one clock cycle away. (This is achieved by inserting message relays on longer links when using the torus topology). The receiver notifies the sender on the subsequent clock cycle whether or not the message was accepted. If the message was accepted, then the sender can simply clear the buffer. If the message was rejected, then the message can be read out of the buffer and placed back in the output queue. When the output queue and the buffer queue both fill up, then the sender must stop producing additional messages until the network clears up.

2.4. Routing

One of the requirements of dataflow networks is that they need to perform reasonably well in the presence of the faulty links and sudden bursts of inter-cluster traffic. However the current WaveScalar architecture uses dimension order routing to route packets between any two clusters in the network. In dimension order routing packets between two clusters are routed along a single predetermined route. Each cluster is given a *x* and *y* coordinate corresponding to its location in the network. Now when a packet is to be routed between two packets it is first routed along the x-dimension given by its row of origin until the packet's *x* coordinate is aligned with the destination cluster's *x* coordinate. Once this step is achieved the packet is routed along the y-dimension until the packet reaches it destination.

Figure-2a shows the union of routes that connect the clusters in row-0 to clusters in column-7. The intensity of the color used to draw a physical-channel depicts the number of routes passing through that channel. A large fraction of the routes between these clusters pass through the links in the top-right corner of the network. Such uneven distribution of routes creates congestion points in the network during heavy traffic, which results in reduced network throughput. Dimension order routing also does not provide the necessary fault tolerance. Figure-3a shows that severing a single link in the grid can partition the network. On the other hand, dimension order routing is easy to implement, ensures deadlock free communication, and routes packets along the shortest route for a given cluster pair.

2.4.1 *Adaptive routing*

To improve the network performance and allow for graceful degradation in the presence of faulty links we decided to replace WaveScalar's dimension order routing with the deadlock free adaptive routing protocol proposed in [2]. Adaptive routing has many advantages over dimension order routing for dataflow networks. Like dimension order routing, adaptive routing routes packets deterministically along the shortest routes when the load in the network is low. However adaptive routing allows packets to be routed along increasing longer routes instead of waiting on network resources during periods of high congestion. This behavior allows adaptive routing to route packets around congested or faulty links thus increasing network performance under heavy traffic and providing a mechanism for graceful degradation when links begin to fail.

The adaptive routing algorithm consists of two main components: a selection function, which distributes the load evenly over the network, and a queuing function that ensures deadlock-free communication. Each packet has a field called '*dimension reversal'* which notes the number of times a packet has been routed in a direction that dimensional order routing would not have chosen.

2.4.1.1. *Selection Function*

The selection function helps a node determine to which one of its neighboring nodes it should route a given packet. In dimension order routing the selection function is a static function. With adaptive routing, we would like our selection function to route packets based on the congestion level in the neighboring clusters. The following algorithm describes our selection function –

- *Create a list of all possible directions – i.e. N, S, E, and W.*
- *From this list remove the incoming direction of the packet.*
- *Order the remaining directions in ascending order of their distance from the destination cluster.*
- *Query the clusters in the given order for availability of adaptive routes.*
- *If an adaptive route is found, note any dimension reversals in the packet header and route the node to its intended destination.*
- *If no adaptive route was found, the packet from here on is routed using only dimension order routing.*

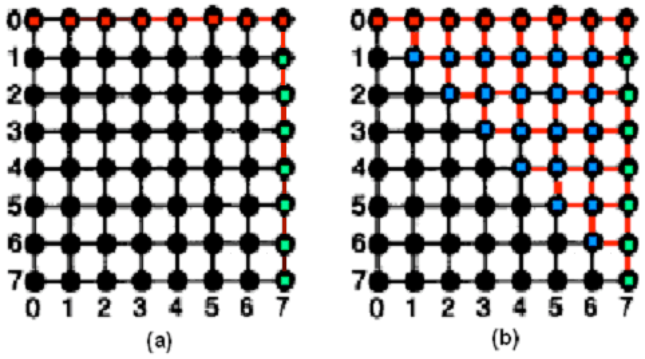

Figure 2. Routing packets between clusters in row-0 and column-7 in an 8 x 8 grid. (a) Dimension order routing created congestion in the top-right corner of the grid. (b) Adaptive routing avoids congestion routing around high load channels.

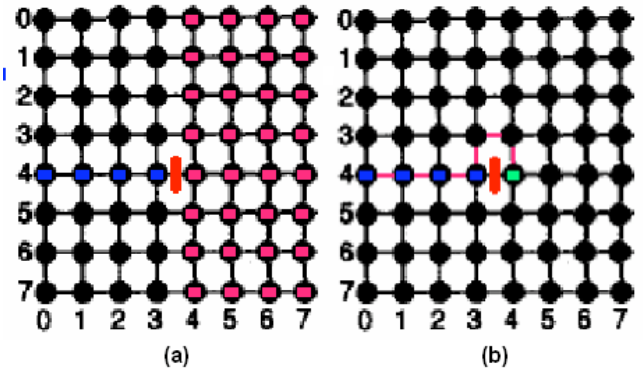

Figure 3. Routing packets in networks with broken physical channels. (a) With dimension order routing, the blue clusters will not be able to send any packets to the pink clusters. (b) Using adaptive routing the network is able to route packets around the broken link.

2.4.1.2. *Queuing Function*

The queuing function avoids deadlocks by ensuring that no packet in the network is waiting on a packet that is in turn waiting on it. Each cluster divides its virtual channels into two distinct classes: adaptive and deterministic. Messages queued in the deterministic channel are routed using only dimension order routing (though this can change in the presence of a known failure in the deterministic path). Messages in the adaptive channels are routed in the direction chosen by the selection function.

When a clustered is queried for the availability of an adaptive route it checks it to see if an adaptive channel exists such that no packet queued in this channel has a '*dimension reversal*' number lower than or equal to the dimension reversal number of the packet that initiated the query.

The algorithm described above is proved to be deadlock free, and remains deadlock free for dataflow networks given the modifications proposed in Section 2.2.

3. Sensitivity Analysis

In order to find the optimum configuration for the network, we developed a cycle-accurate network simulator that used dataflow traces from WaveScalar to evaluate the effect of different configurations in overall performance. The simulator allows us to evaluate a wide variety of configurations, and a wide variety of applications. Configurable parameters include the topology, routing algorithms, number of nodes, network bandwidth, network queue sizes, number of channels, and the number of packets per message.

The baseline configuration for all simulations is a 4x4 grid of nodes and a grid network topology using static dimension-order routing.

The simulations were run using traces of 2 billion instructions from five different benchmark applications: *art* is a Spec2000 floating point benchmark, *mcf* is a Spec2000 integer benchmark, *lu* is a Splash 2 multithreaded kernel benchmark, *ocean* is a Spash 2 multithreaded application, and *fir* is a custom dataflow microkernel. These applications represent a spectrum of different types of applications that a dataflow processor could be expected to run. It is important to note that only dataflow configurations that implement wave-ordered memory [1] can run the Spec2000 and Splash 2 benchmarks.

The metric used to evaluate the performance of the network is the total number of cycles required to complete execution of the trace. This is equivalent to the actual performance of the network, though it does not take into account the performance of the underlying computational structure. Other available metrics are the max and average time through the network, the max and average number of hops, and the max and average number of routing failures. All of these metrics show the same general shape in the curves, and so we use the simulation runtime to represent all of these metrics.

All of the simulations in this study allow only one parameter to change at a time to make the effect of changing that parameter clear. In future work, we would like to vary multiple parameters at a time and evaluate whether there is constructive or destructive interference between multiple parameters.

3.1 Link Bandwidth

The link bandwidth is equivalent to the number of messages that can be sent in a single direction from one node to an adjacent node in a single clock cycle. This is a critical question since it seems to be the most obvious way to handle the bursty traffic patterns in dataflow networks. We ran simulations with bandwidths from 1 to 16 messages per cycle, and an equal number of infinitely long channels ensure that there was no contention for queues when the links were fully utilized. Each benchmark was run in on both the grid and the torus topologies. The results were normalized to the performance for a bandwidth of one message per cycle, and are shown in Figure 4.

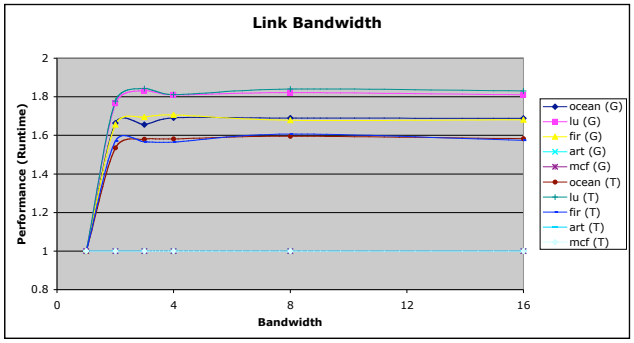

Figure 4: Link bandwidth performance sensitivity

The graph shows that there is a significant performance benefit when increasing the number of links to 2, but very little performance benefit beyond that. This is a surprising result given the bursty nature of the dataflow traffic. There are two possible explanation: (1) quickly handling the bursts of network traffic is not essential to overall performance, or (2) there is another bottleneck that is limiting performance. There is some evidence for (1) in that only a very small percentage of the traffic has a fan out of more than two destinations. In most workloads, fewer than 3% of the traffic is bursts from one particular node to a large number of other nodes. It is possible that the occurrences of high-fanout instructions are not critical to overall performance.

The results for art and mcf seem to indicate that their performance does not change. This is anomalous, however, since these particular benchmarks have very little inter-node communication during their first 2 billion instructions. The benchmarks do exhibit significant amounts of inter-node communication, but simulations long enough to capture this traffic were not available in time for this publication.

3.2 Queue Length

Another important parameter is the queue length. For these experiments, we varied the queue length of both the channels in the configuration. The bandwidth was set to equal the number of channels to ensure that each queue could potentially accept a message on every cycle. Experiments were run for queue sizes ranging from 1 to 64 entries. The results are shown in Figure 5.

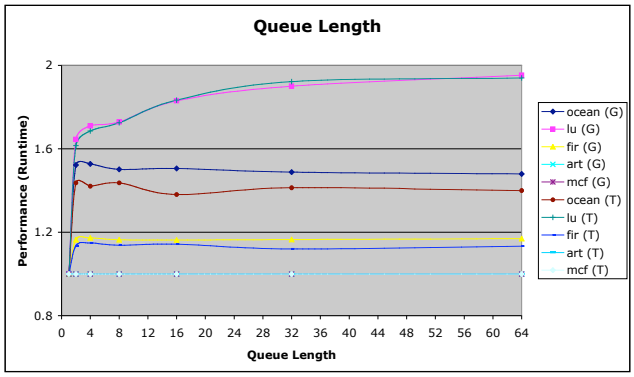

Figure 5: Queue length performance sensitivity

The performance of the network seems to improve dramatically from a one input queue to two input queues, but there seems to be relatively little performance gain by adding additional queue entries. Only *lu*, the most bursty of the benchmarks, seems to benefit significantly from larger queue sizes.

The results for queue size are somewhat biased by the fact that the trace simulator does not model contention within each node for network resources. The trace simulator makes the assumption that each message can be accepted once it reaches its destination node, and successfully passes through the input queues. If the nodes were less likely to accept incoming messages, then there would likely be improved performance for longer queue sizes.

One other reason that the optimal queue size is so small is that dataflow placement tries to concentrate instructions in nearby locations. When instructions communicate across nodes, they tend to take the same path. As long as there are no other paths competing for the same link, then there is likely to be nothing preventing the input flow from draining. In fact, many placement algorithms take into account the effect of placement on network communication patterns, and hence try to prevent those patterns from crossing. Our studies used the exp4 placement algorithm [1] which is one of the algorithms that takes into account this behavior.

3.3 Virtual Channels

In the current WaveScalar architecture the buffer space available to each physical channel is implemented as a single queue. This design choice couples the buffer resources of the network with its channel resources, thus resulting in decreased network throughput.

At the hardware level, only the packet at the head of the queue can be dequeued and routed onto the physical channel in one clock cycle. This means that if a packet at the head of the queue is blocked waiting for a network resource to be freed, all other packets queued behind this packet will also be blocked even if they do not need access to the unavailable resource. For example, in Figure 6 Packet A

arrives before Packet B and is therefore queued ahead of B in the single buffer queue. Unfortunately for Packet B, Packet A is blocked because the destination node it is trying to reach is currently busy. Packet B on the other hand has no such dependencies but is blocked nonetheless, resulting in an idle physical channel and lowered network throughput.

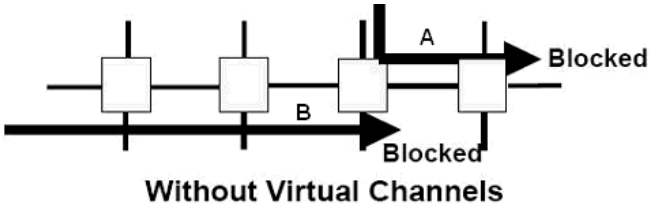

Figure 6.

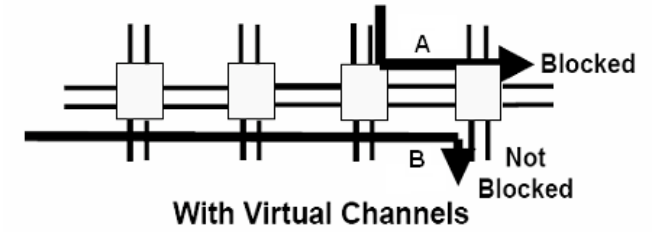

Figure 7.

To circumvent this problem we implemented Virtual-Channel based flow control proposed by W.J. Dally [3]. Virtual-Channels increase network throughout by decoupling buffer and channel resources. Instead of allocating the available buffer space to a single queue, the buffer space is divided among several smaller queues or virtual channels. When using static routing, incoming packets are placed into the queue with most available space. In a given clock cycle every virtual channel is capable of placing the packet at the head of its queue onto the physical channel. Fairness is ensured by using a round-robin scheduling scheme. The scenario depicted in Figure 8 is recreated in Figure 7 for a network that uses virtual channels. In this network Packet B is no longer blocked due to Packet A, resulting in increased network throughput.

This raises the question of how many virtual channels are needed for a dataflow network. At one extreme, a buffer could be divided into N virtual channels that can hold 1 packet each. However every additional virtual channel involves additional hardware complexity, which comes at the cost of reduced performance per area. Also, the performance gain due to virtual channel tends to level off after enough virtual channels have been added to the network making the marginal gains from additional channels unnecessary in lieu of their hardware cost.

We determined the ideal number of virtual channels for the WaveScalar network by running simulations with various numbers of virtual channels. We used the baseline configuration of 2 links per channel, but assigned a random time that each packet had to remain in the queue until it was eligible to be routed. This modeled congestion, where a greater number of virtual channels is helpful. The results of the simulation are shown in Figure 8.

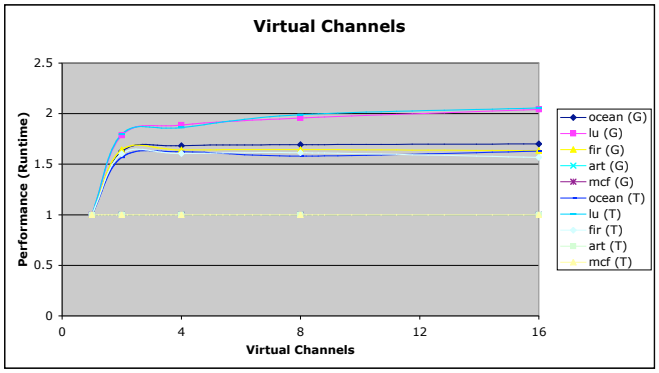

Figure 8: Virtual channel performance sensitivity

The graph shows that the performance increase for the number of virtual channels levels off after 2 to 3 virtual channels. This is not surprising since there is not much reason for packets to block in the current network. In the face of increased contention, additional virtual channels may improve performance, but for these typical loads, there is not enough contention to make numerous virtual channels worthwhile.

3.4. Link Width

The baseline configuration for the WaveScalar architecture uses message links that are wide enough to carry the entire message in a single cycle. We have carried this assumption through the other studies, but the assumption should be evaluated to ensure that it is necessary to have these very wide busses.

For these simulations, the data bus size (which we chose to be 128-bits, in accordance with the WaveScalar bus size) was cut by factors of 2. The overhead associated with recombining the messages at the destination was ignored. The results for the experiment are shown in Figure 9.

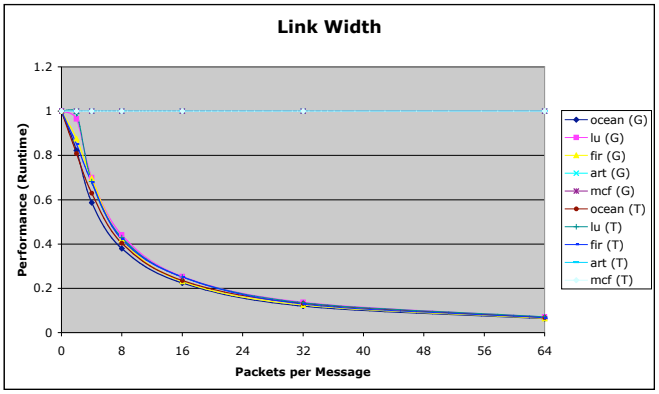

Figure 9: Link width performance sensitivity

As the graph shows, performance goes down very quickly as the number of packets per message increases. By cutting the bus width by 2, performance degrades by 20%. Cutting the data bus width by 4 degraded performance by more than a 40%. Clearly, it is critical that the network links pass full messages in parallel.

3.5. Adaptive Routing

The performance impact of adaptive routing was studies relative to the same sensitivity analysis experiments run previously. The results are shown in Figures 10, 11, and 12. The net result is that adaptive routing provided a between a 1.5x and 2x speedup on most applications, which shows adaptive routing to be essential not only for fault tolerance, but also for overall performance.

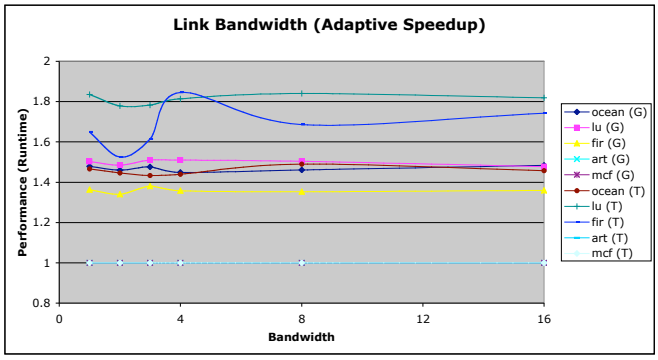

Figure 10: Speedup of link bandwidth sensitivity analysis using adaptive routing.

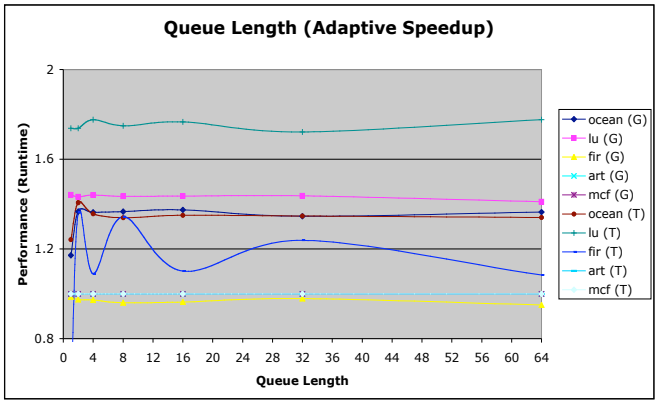

Figure 11: Speedup of queue length sensitivity analysis using adaptive routing.

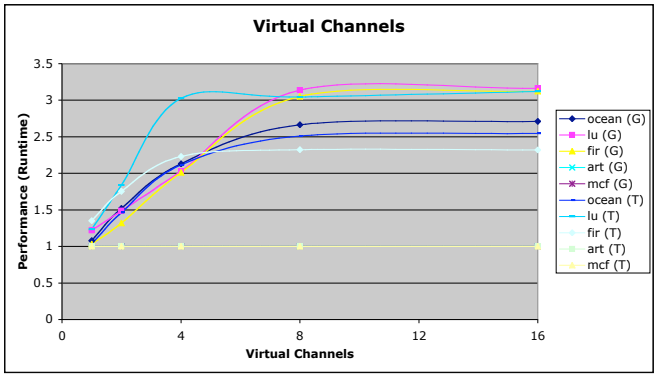

Figure 12: Speedup of virtual channel sensitivity analysis using adaptive routing.

4. Area Analysis

In typical network applications, the physical size of the network switches and links is of little concern. For on-chip networks, the area is a primary concern. The economic viability of on-chip systems is directly tied to the total die area required to implement the design. Users demand high performance as well as small area, so finding the proper balance between the two is critical to making a high-performance yet economically viable design.

On-chip networks also have very tight timing constraints, significantly tighter than in typical networks. Network transfers must take place within integer multiples of the system clock cycle. If the signal propagation time along one link in the network is longer than the propagation time along other links, then the designer must choose whether to (1) slow the transfer rates of all the other links to make a common link propagation time, or (2) add additional logic to the routing and message handling mechanisms to consider the latency of the link before using slower links. There is a balance here as well between using more complicated routing decisions that will take advantage of the different link times, yet take time themselves to determine the correct path and thus reduce throughput.

4.1 Area Metrics

In order to determine the optimal balance, of performance and area we need a metric that expresses performance as a function of area. For simplicity, we choose instructions per cycle (IPC) per square millimeter, or IPC/mm^2. For performance-critical designs, IPC can be multiplied by a constant, or even raised to a small exponent. The same could be done to the area component for area critical designs.

When determining the effect of the network links, using the chip area is questionable since the links are actually implemented in metal layers that sit above the base layer, and hence do not contribute to the actual die area. Instead, the viability of the links is measured in a somewhat subjective metric called degree of congestion. Too many

links or links that are routed poorly tend to cause congestion, which requires links to be routed in higher and higher layers of metal. This is acceptable as long as there are available metal layers. When metal layers are no longer available, the design cannot be implemented. This makes a rather binary metric, which causes us to completely disregard a design if it cannot satisfy the congestion criteria. While this may seem excessive, unimplementable designs are of little value, and hence should not be considered further given current technological constraints.

4.2 Synthesizable Model

In order to get the area of the design, we implemented an RTL-level model using SystemC. From this synthesizable model, we used Synopsys DesignCompiler and DesignCompiler Ultra for logical synthesis.

The design rules for manufacturing devices have undergone dramatic changes at and below the 130nm technology node [19]. Issues such as crosstalk, leakage current, and wire delay have required synthesis tool manufacturers to upgrade their infrastructures. The data we present in later sections is derived from the design tools specified by Taiwan Semiconductor Manufacturing Company's TSMC Reference Flow 4.0. TSMC selected these tools specifically to handle the increased complexity of 130nm and smaller designs. By using these up-to-date tools and technology libraries, we ensure, as best as possible, that our results scale to future technology nodes. As a result of the new design rules, designs at and below 130nm are extremely sensitive to placement and routing. Therefore, we did not use the area or delay numbers that are produced after logical synthesis by DesignCompiler. Instead, we fed its generated netlist into Cadence Encounter for floorplanning and placement, and then used Cadence NanoRoute for routing. After routing, we extracted the appropriate area values. We used Encounter to extract the resistance and capacitance values, and produce the net timing information. This information was then fed into Synopsys PrimeTime for static timing analysis.

4.3 Area Results

Table 1 shows the results of area for various configurations of a single network switch. The baseline configuration is 2 links, 2 channels, and a queue length of 4. The table shows the effect of on area of changing (a) the number of virtual channels, (b) the link bandwidth, and (c) the queue length per channel. Doubling the queue length is the most expensive choice as it nearly doubles the total area of each network switch. Increasing the number of channels similarly increases area. Increasing bandwidth has less of an effect on area, but increases congestion

4.3.1 Performance per Area

Using these area results, we can go back to the performance results from the previous sections to evaluate the performance per unit area. Graphs of performance per unit area are shown against queue length, number of virtual channels, and link bandwidth in Figures 13, 14, 15. The figures all show sharp peaks around 2 for each of the parameters. This leads us to conclude that the optimum configuration for both performance and performance per unit area is 2 channels, a bandwidth of 2, and a queue length of 2.

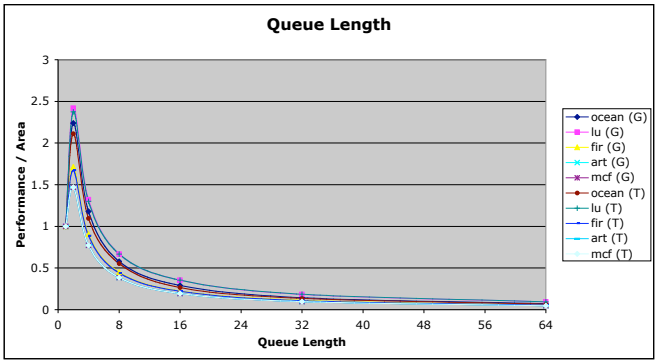

Figure 13: Performance per unit area for queue length sensitivity

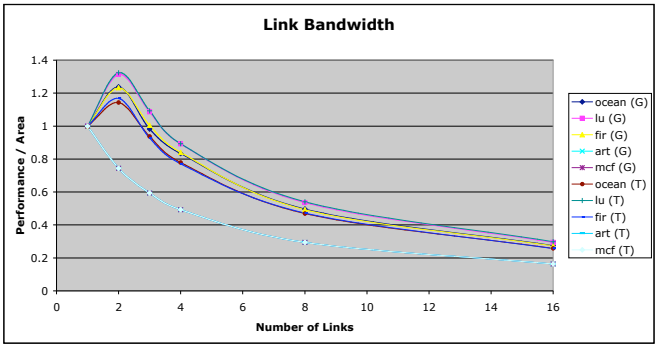

Figure 14: Performance per unit area for link bandwidth sensitivity

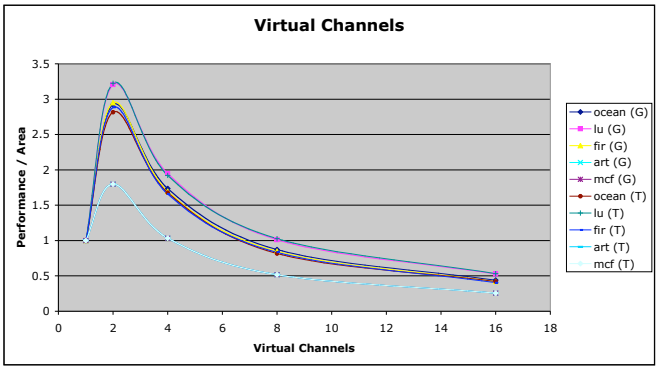

Figure 15: Performance per unit area for virtual channel sensitivity

4.3.2. Timing Results

The only two topologies that were implemented in the RTL model are the grid and the torus. In the grid topology,

the links are all the same length, and have approximately the same delay. The longest link delay for the grid topology is 2.76 ns. For the torus topology, the different links have different latencies. The worst case latency is 6.16 ns for a 4x4 grid. Since sending a message through the 4xx4 grid would require 3 hops at 2.76 ns per hop, the long link on the torus gets messages to far nodes about 34% faster than going through the network, and helps keeps long-distance traffic out of the central grid. The scalability of the torus network is the topic of future work, but initial estimates

Table 1: Area values for network switch with various number of channels, bandwidth, and area

Channels	Area (mm^2)
1	143064.8
2	189387.6
4	282033.2
8	467324.4
16	837906.8

Bandwidth	Area (mm^2)
1	141016.6
2	189387.6
3	237758.6
4	286129.6
8	479613.6
16	866581.6

QLength	Area (mm^2)
1	97277.9
2	143064.8
4	189387.6
8	370586.4
16	741164.8
32	1482329.6
64	2964659.2

suggest that this slight performance advantage will hold for a wide range of configurations.

5. Conclusion

Our study has shown that high-performance dataflow networks can be implemented in a reasonable area. The optimal configuration is small, consisting of 2 channels, 2 links, and 2-4 entries per queue. Also, adaptive routing proved to be not only important for fault tolerance reasons, but it significantly improves overall performance. Also, there is little difference between the grid topology and the torus topology in the majority of benchmarks, though the torus topology gained a slight edge when coupled with adaptive routing and a heavily network-centric workload.

6. Acknowledgements

We would like to sincerely thank Martha Mercaldi for writing the trace gathering tool. We would also like to thank Steve Swanson for his help in introducing us to the WaveScalar simulator. Thanks also to Andrew Peterson, Andrew Schwerin, and Mark Oskin for their timely help with the computing resources. Finally, we would like to thank Mike Scott for his help and guidance while implementing the RTL design.

Bibliography and Related works:

[1] Swanson, S., Michelson, K., Schwerin, A., Oskin, M. "WaveScalar". 36th Annual International Symposium on Microarchitecture (MICRO-36). Dec 2003

[2] Dally, W.J. "Virtual-Channel Flow Control". IEEE Transactions on Parallel and Distributed Systems 3, No. 2, 194-205. 1992

[3] Dally, W.J., Seitz, C.L. "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks". IEEE Transactions on Computers Vol C-36. No. 5. May 1987

[4] Dally, W.J., Towles, B. "Route Packets, Not Wires: On-Chip Interconnection Networks". DAC 2001. June 2001.

[5] Agarwal, V., Hrishikesh, M.S., Keckler, S.W., Burger, D. "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures" 27th Annual International Symposium on Computer Architecture (ISCA-27) May 2000

[6] Taylor, M.B., et. al. "The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs", IEEE Micro, Mar/Apr 2002.

[7] R. Nagarajan, K. Sankaralingam, D. Burger, and S.W. Keckler. "A Design Space Evaluation of Grid Processor Architectures". 34th Annual International Symposium on Microarchitecture (MICRO), pp. 40-51, December, 2001

[8] Dally, W.J. "Deadlock Free Message Routing in Multiprocessor Interconnection Networks" IEEE Trans. Compt., vol C-36, no. 5, pp 547-553, May 1987

[9] The MOSIS Service "MOSIS Scalable CMOS Design Rules" Revision 8.00. http://www.mosis.com/Technical/Designrules/scmos/scmos-main.html. Accessed 11/07/04.

[10] Singh, A., et. Al. "GOAL: A Load-Balanced Adaptive Routing Algorithm for Torus Networks". ISCA '03.

[11] Ramany S., and Eager D. "The Interaction between Virtual Channel Flow Control and Adaptive Routing In Wormhole Networks". ICS –94.

[12] Even S., and Kupershtok R. "Layout Area of the Hypercube". Journal of Interconnection Networks – 2003.

[13] J.R. Gurd, C. C. Kirkham, and I. Watson. "The Manchester Prototype Dataflow Computer". Communications of the ACM – 1985.

[14] P.E. Berman, L. Gravano, and P.D. Gustavo. "Adaptive Deadlock and Livelock-Free Routing with Minimal Paths in Torus Networks". SPAA '92.

[15] W. J. Dally. "Performance Analysis of k-ary n-cube Interconnection Networks". IEEE 1990.

[16] A. Patel, A. Kusalik, and C. McCrosky. "Area-Efficient VLSI Layouts for Binary Hypercubes". IEEE Transactions on Computers, 2002.

[17] Granacki, J. and Vahey, M., "Monarch: A High Performance Embedded Processor Architecture with Two Native Computing Modes." High Performance Embedded Computing Workshop, September 2002.

[18] Keynote Address: John Shen, Intel Corporation. MICRO-37

[19] Semiconductor Industry Association, "International Technology Roadmap for Semiconductors 2003 - Executive Summary", 2003

[20] P. K. Howard. "Incomplete Hypercube". IEEE Transactions on Computers, 1988.

[21] L. Benini and G. D. Micheli. "Networks on Chips: A New SoC Paradigm", IEEE Computer, 2002

[22] A. Martin, "The TORUS: An exercise in constructing a processing surface", Proceedings of the VLSI Conference, 1981