In Proceedings of *Hot Interconnects '93*, August 5-7, 1993, Palo Alto, California

# A Cost and Speed Model for k-ary n-cube Wormhole Routers

**Andrew A. Chien**
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Avenue
Urbana, IL 61801
Email: *achien@cs.uiuc.edu*

## Abstract

A great deal of research has been published on the performance of wormhole routers with advanced features such as adaptivity and virtual lanes. In most cases, the effectiveness of such novel routers is evaluated on the basis of the achieved network throughput (channel utilization), ignoring the important effects of implementation complexity. In this paper we describe a parameterized cost model for router performance, characterized by two numbers: router delay and flow control time. Grounding the cost model in a 0.8 micron gate array technology, we use it to compare a number of proposed routing algorithms.

Based on these design studies, several insights regarding the implementation complexity of adaptive routing are clear. First, header update and selection is expensive in adaptive routers, suggesting the absolute addressing should be reconsidered. Second, virtual channels are expensive in terms of latency and cycle time, so decisions to include them to support adaptivity or even virtual lanes should not be taken lightly. Third, requirements of larger crossbars and more complex arbitration cause some increase in the complexity of adaptive routers, but the rate of increase is small. Finally, the complexity of adaptive routers significantly increases their setup delay and flow control cycle times, implying that claims of performance advantages in channel utilization and low load latency must be carefully balanced against losses in achievable implementation speed.

## 1 Introduction

Routing networks are critical to the performance of parallel machines because they determine the efficiency with which processing elements can communicate and cooperate. Three fundamental characteristics of a network are its topology, routing and flow control policies. We consider a family of networks which have k-ary n-cube topologies, allow adaptive or multipath routing (i.e. choose paths dynamically based on router status), and implement wormhole routing. We focus on examining the increased implementation complexity due to adaptive routing with a particular emphasis on how these changes affect achievable router speed. Our evaluation focuses on two metrics of router implementation speed: setup delay and flow control cycle time. These two metrics directly affect two critical dimensions of router performance, latency and bandwidth.

Many studies have been published reporting router performance in network clock cycles, assuming unit router delay (or some other arbitrary number of cycles). But, because the network clock cycles for different routers will generally not be the same, results from these studies cannot be compared directly. In this paper, we develop a parametric cost and speed model for a family of routers based on a canonical architecture. This model allows us to evaluate and compare the cost of various routing features in hardware resources as well as achievable router speed.

Making concrete comparisons requires a concrete cost and speed model. We instantiate our parametric model, using a 0.8 micron gate array technology and produce specific constants for all aspects of the model. This model is then used to compare a selection of routing algorithms: dimension order [14], planar adaptive [8], turn model [28], and *-channels [6]. The comparison produces a variety of insights about the relative latency and achievable clock rate of these routers. In particular, we find that all of the adaptive routers we consider are significantly slower (longer setup latency and lower clock speeds) than deterministic routers due to header selection cost and the requirement of virtual channels for deadlock prevention. The negative performance impact of virtual channels is particularly severe, incurring setup latency and reducing network clock speeds. Consequently, a designer of routing networks must carefully balance the benefits of adaptive routing against these significant costs in deciding whether or not to include adaptivity.

## 2 Background

High performance routing networks, the subject of significant study over the last fifteen years, are currently in widespread use in machines such as the Intel

Paragon [9], Intel iWarp [7, 30], Ncube/2 [26], and the MIT J-Machine [11, 13]. All of these multicomputer systems use *direct networks*, meaning that the computing nodes are embedded in the network topology, and as a result, some nodes are closer than others. In addition to use in multicomputers, direct networks are gaining acceptance in shared memory machines such as the MIT Alewife [2, 1], Stanford DASH [23], and Tera Computer's TERA machine [3]. All of these machines use topologies from the family of k-ary n-cubes. That is, in systems with $k^n$ nodes, the networks have n dimensions and k nodes along each dimension. All of the routers considered in this paper can be used in k-ary n-cube topologies.

## 2.1 Wormhole Routing

The performance of parallel systems is closely tied to the throughput and latency provided by their routing networks. Many of these networks use virtual cut-through [21] or *wormhole routing* [10], a technique which reduces message latency by pipelining its transmission over a number of channels along its path. For the remainder of this paper, we focus on wormhole routed, direct networks.

The important distinction between virtual cut-through routing and wormhole routing is the action taken when a message's path is blocked. In virtual cut through, the head of the message is stopped and the rest of the message continues to move. Eventually, if the head is stopped for long enough, the entire message arrives and is buffered at the router where the head is blocked. One major advantage of virtual cut through is that deadlock prevention is relatively easy because a blocked message does not tie up any channels, only buffers. A major disadvantage is that routers must be able to buffer a number of entire messages, increasing router size and perhaps reducing router speed.

In contrast, a wormhole router stops a message "in place" when its head is blocked. The head of the message stops, and the remainder of the message is stopped, holding the buffers and channels along the path it has already formed. The primary advantage of wormhole routers is that router buffer requirements can be small, allowing routers to be extremely small and fast. The obvious disadvantage is that by allowing a message to retain the buffers and channels, wormhole routing increases the possibility of resource cycles which cause deadlock. A number of schemes have been developed which solve this problem [10, 28].

## 2.2 Adaptive Routing

Adaptive routing has been proposed as a means of improving network throughput and performance robustness. A wide variety of adaptive routing algorithms have been proposed [24, 12, 16, 8, 28, 6]. However to date, most evaluations of adaptive routing algorithms use register transfer level simulation which shows how adaptive routing increases channel utilization but cannot measure the impact of additional router complexity on network latency and bandwidth.

Adaptive routing allows paths to be chosen dynamically based on information about which channels are available, potentially reducing network latency and increasing network bandwidth by making better use of network resources. However, while adaptive routing increases routing freedom, it also increases the cost of deadlock prevention. Significant additional hardware may be required (as much as $2^{n-1}$ virtual channels for fully adaptive routing in n-dimensional networks [24]). Further, adaptive routing increases the complexity of routing decision logic and crossbar size. Both of these costs can reduce router speed, and thus must be weighed against the benefits of adaptive routing to determine how much adaptivity and which routing algorithm is most attractive. Concerns about the complexity of adaptive routers are not new, as a number of researchers have proposed limited adaptive routing schemes which reduce both routing freedom and hardware complexity [8, 16, 28]. Several examples of such routing algorithms are studied in Section 5.

# 3 Basic Router Functions

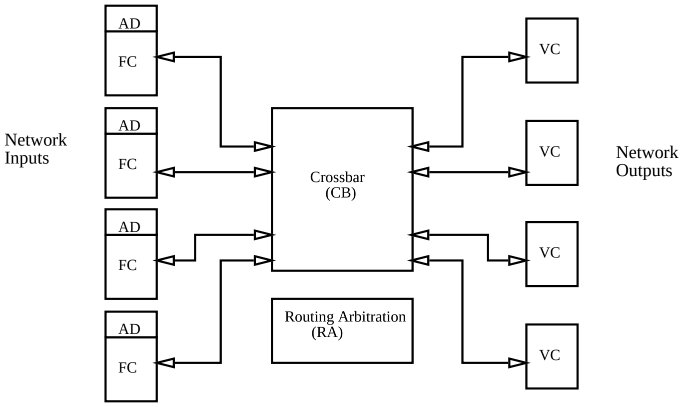


**Figure 1**: A canonical architecture for a wormhole router, data and flow control paths only.

A wormhole router must perform several basic functions: switching, routing, flow control, multiplexing physical channels, inter-chip signalling and clock recovery. A basic wormhole router architecture is shown in Figure 1. Data moves from left to right through the router and the complementary flow control signals move along parallel paths in the opposite direction. The essential components of the router and their functionality are described below.

- **Crossbar** (CB) provides the basic switching function from router inputs to outputs. If the switching function is not complete (all inputs to all outputs), sometimes the crossbar can be partitioned into a set of smaller crossbars for higher performance and lower cost. The size of the crossbar may be proportional to the number of inputs, outputs, and virtual channels.

- **Flow control units** (FC) perform flow control between routers, stopping a message in place in the network, if necessary. This function requires control logic and some buffering. Signalling flow control information across the channel incurs delay, so sufficient buffering to capture the data in flight on the channel must be available.

- **Address Decoders** (AD) examine the packet header and generate the set of possible routes, based on the routing algorithm. This can range from a trivial bit selection to several arithmetic operations. Address decoders also compute the updated header. Most routers use relative addressing.

- **Routing Arbitration Logic** (RA) chooses the path and connects and disconnects the input to an appropriate output based on the routing algorithm and network status information. It must do arbitration for the router outputs, so its complexity and latency increase with the number of choices possible for packet (routing freedom).

- **Virtual Channel Controllers** (VC) multiplex the physical channels, providing a set of virtual channels that can progress independently. The controller complexity increases superlinearly in the number of virtual channels, due to increased buffering and arbitration requirements. Physical channel arbitration is on the critical path for router latency.

Packets arriving at the router inputs are fed into the address decoders which generate a set of requests for possible outputs. The routing arbitration logic combines the requests from all of the inputs and the router status and matches inputs to appropriate outputs.[1] If no appropriate outputs are available, an input may be blocked. Our router architecture can connect all inputs to outputs in a single cycle (assuming no output conflicts). However, as network packets are typically multiple flits, this capability is not essential, making connections sequentially does not significantly reduce performance.

Once an appropriate output has been selected, the switch connection is maintained for the entire packet. Following the last flit, the switch connection is broken and the output freed for subsequent communications. While no single router architecture is appropriate for all routing algorithms and possible design parameters, this basic architecture is attractive for a range of routing algorithms [14, 8, 28] and has in fact been used for a number of practical routers [14, 15, 29, 4, 31, 9]. Our canonical router architecture captures an attractive design for networks of modest dimension and modest numbers of virtual channels, providing a basis for direct comparisons amongst a set of routing algorithms. Beyond three or four dimensions and about four virtual channels, requiring a full crossbar for all virtual channels at both inputs and outputs can become a significant design consideration. Beyond this point, memory-based or bus-based approaches may be more attractive.

### 3.1 Performance Metrics

Routing network performance has two important attributes: routing latency (for short messages or on large machines this is dominated by the time to set up a connection) and bandwidth (determined by the flit size and the time to do a flow-control operation). A router contributes to these times through its routing setup latency, and its flow control latency, respectively. Other contributions include channel delay (time of flight) and clock recapture overhead. Because these inter-router times depend primarily on topology and packaging, not routing algorithm, they are essentially identical for the router designs we consider. Consequently, we focus instead on the internal router contribution to delays. Characterizing attainable router speeds for a routing algorithm makes it possible to scale simulation studies of channel utilization or latency in clock cycles, providing a basis for normalized comparisons.

**Routing Latency** The per-node routing latency consists of two parts: the inter-router delay and the internal router latency. We focus on the internal router

[1] For some routing algorithms [8, 28], the router state is only used to choose between deadlock-free alternatives, allowing the router state to be approximate.

| Module | Parameter | Gate count | Delay |
|---|---|---|---|
| Crossbar | P (ports) | $O(P^2)$ | $c_0 + c_1 * \log P$ |
| Flow Control Unit | none | $O(1)$ | $c_2$ |
| Address Decoder | none | $O(1)$ | $c_3$ |
| Routing Decision | F (freedom) | $O(F^2)$ | $c_4 + c_5 * \log F$ |
| Header Selection | F (freedom) | $O(F)$ | $c_6 + c_7 * \log F$ |
| VC Controllers | V (# vc's) | $O(V)$ | $c_8 + c_9 * \log V$ |

**Figure 2**: Gate Counts and Delays for the Router modules. P is the number of inputs or outputs for the crossbar. F (freedom) is the routing freedom, the number of output choices an input can have, and is typically the same as P. V is the number of virtual channels that a virtual channel controller must multiplex onto the physical channel.

latency — the time to create a connection through a router — which can be decomposed into the following contributions:

1. Decode Address and generate connect requests ($T_{AD}$)
2. Arbitration ($T_{ARB}$)
3. Updated header selection ($T_{SEL}$)
4. Drive data through the crossbar ($T_{CB}$)
5. Virtual channel controller delay ($T_{VC}$)

The speed of each of these operations directly affects the router latency. Precisely how these factors combine depends on the degree of routing freedom, how arbitration for resources occurs, and how deadlock prevention is achieved. These issues are examined in detail in a series of specific routing examples in Section 5.

**Flow Control Latency** A router's channel bandwidth depends on the size of a flow control unit (flit) and the time required to do a flow control operation. The internal flow control latency limits the flit rate on network channels. Though higher bandwidth can be achieved by increasing the flit size, such a choice increases router complexity by introducing separate logical (flow control) and physical signalling rates, requiring additional buffering, slowing backpressure, and increasing routing latency. In contrast, equating flit and physical transfer unit (phit) size allows a simple token passing or XON/XOFF (stop/start) protocol, simplifying router design. Equating the flit and phit sizes ties the flow control delay to the signalling rate.

Flits are units of resource multiplexing (internally and externally), so flow control latency determines the network's ability to share internal connections and external channels amongst different packets. The unit of multiplexing directly affects the responsiveness of the network. In addition, flow control speed determines the amount of buffering needed.

The critical path for flow control operations can be broken down into the following steps:

1. Flow controller delay ($T_{FC}$)
2. Forward crossbar switch delay for data ($T_{CB}$)
3. Virtual Channel Controller delay ($T_{VC}$)

Again, the particular contribution of these module delays to overall flow control latency depends on the choice of routing algorithm. No backward propagation delays are included because the flow control is fully pipelined. Typically, the main factor in flow control delay is whether or not virtual channels are needed for deadlock freedom. Surprisingly, virtual channel controllers can incur significant delay, contributing a large fraction of the total flow control delay. We examine the specific contributions in the context of several routing algorithms in Section 5.

## 4 A Parametric Cost Model

A parametric cost model based on the complexity and speed of each hardware module allows us to compare the achievable hardware performance (complexity) of a variety of router designs. In this section, we characterize the complexity of each module in our canonical router architecture in size and delay, examining their scalability for larger and enhanced router designs.

### 4.1 Basic Module Delays

A wide variety of routing algorithms can be implemented with our canonical router architecture and slight variants of the following basic modules. The basic functionality of the modules does not vary much

| Module | Parameter | Gate Count Formula | Typical Size |
|---|---|---|---|
| Crossbar | P | 29 $P^2$ | 460 (P=4) |
| Flow Controller | n.a. | constant | 320 |
| Address Decoder | n.a. | constant | 100 |
| Routing Decision | F | 17 $F^2$ | 280 (F=4) |
| VC controller | V | 126 V | 380 (V=3) |

**Figure 4**: Sample Gate Counts for Router Modules. All gate counts assume 16 bit flits and 16 bit wide internal data paths.

| Constant | Value |
|---|---|
| $c_0$ | 0.4 |
| $c_1$ | 0.6 |
| $c_2$ | 2.2 |
| $c_3$ | 2.7 |
| $c_4$ | 0.6 |
| $c_5$ | 0.6 |
| $c_6$ | 1.4 |
| $c_7$ | 0.6 |
| $c_8$ | 1.24 |
| $c_9$ | 0.6 |

**Figure 3**: Module Delay constants for a 0.8 micron CMOS process. All values in nanoseconds.

from one routing algorithm to the next. However, there are some subtle changes in logic and critical path dependences. The minor changes necessary are examined in greater detail in Section 5. Here, we describe each router module and its implementation complexity. Formulas for the gate count complexity and delay can be found in Figure 2.

- **Crossbar** While there are many possible implementations, most routers use a tree of gates, or selectors for each output. Thus, crossbar size grows as the product of the number of inputs and outputs, and the delay grows logarithmically due to the depth of the selection trees.

- **Flow control unit** For each channel, a flow control unit manages the local buffers, preventing overflow and underflow. When the number of channels is increased, more flow control units are required, but the complexity of each does not increase.

- **Address Decoder** For each packet, an address decoder extracts the packet address and generates requests for the acceptable outputs, based on the routing algorithm. The address encoding can affect the complexity of this operation, but the encoding is typically chosen to minimize such effects. We model the decoder delay as constant. The address decoder also updates the header, decrementing, incrementing, or passing a dimension address based on the routing decision. While new headers can be calculated in parallel with routing arbitration, header selection is on the critical path and has delay logarithmic in the number of possible routing choices.

- **Routing Arbitration Logic** The arbitration module takes the requested outputs and the switch status as inputs and generates switch connections (output channel assignments). In the worst case, packets might arrive on all inputs simultaneously, requiring circuits with at least logarithmic depth in the number of inputs. We assume simple arbitration algorithms implemented with log depth circuits.

- **Virtual Channel Controller** Each virtual channel controller must include at least one buffer for each virtual channel being multiplexed onto the physical channel. This implies at least a linearly increasing gate count with respect to the number of virtual channels. However, the VC delay increases logarithmically in the number of virtual channels due to arbitration for the physical channel.

### 4.2 An Instantiated Cost Model

Our parameterized model is based on a family of routers designed in the past year [4]. However, because the constants are not explicit, it cannot be used to make evaluations of relative importance of factors to router speed nor for comparisons between routing algorithms. In this section, we instantiate our model for a 0.8 micron CMOS gate array technology, evaluating the relative cost of operations such as routing decision, crossbar setup, and flow control. In Section 5, we use the instantiated model to compare a variety of proposed routing algorithms.

Our performance model is instantiated based on a 0.8 micron CMOS gate array process.[2] For reference, basic 2-input NAND gates in this technology have intrinsic delay below a quarter of a nanosecond and delay of 350 to 750 picoseconds for one to five gate loads (based on typical interconnect routing). Our timing estimates are based on nominal estimates of wiring capacitance, nominal processing, and nominal operating temperature. Using complete designs for each module combined with gate-level timing estimates gives the numbers in Figure 3 for the constants ($c_i$) used in the expressions of delay. In some cases, the models do not match the exact speeds realizable, due to discrete gate choices such as using 2-input versus 3-input gates. Instead, the models are designed to match the basic growth of the latency. Though the constants will differ across hardware technologies, we believe that the essential elements of fan-in, fan-out, and logic complexity that affect the delays in our designs will also be critical factors in other gate arrays or even custom VLSI.

For each router module, we characterize its cost in gates and its speed (latency) as a function of the number of inputs, router dimension, etc. Though large increases in gate count can reduce performance due to increased interconnect delay, gate count is probably not a significant performance factor for the designs considered, most designs fall in the 10,000-30,000 gate range; well within the limits of current technology.

An alternative way to characterize delays is in units of gate delays. However, since such delays are nonuniform due to differing fan-in and fan-out as well as the use of different speed gates in our design, we report the delays in nanoseconds. Module gate counts expressions are shown in Figure 4. Gate counts for entire routers are given at the end of Section 5, in Figure 10. These gate counts cover all of the router internals, but omit counts for off chip connectivity – external clock synchronization and pads.

# 5 Applications

Applying the canonical architecture and performance model to routing algorithms highlights their differences and provides clock rate and latency estimates. Such estimates can be used to normalize comparisons of register-transfer simulation results. In this section, we first consider a baseline, a deterministic router (dimension-order or e-cube), and then compare it to a variety of proposed adaptive router designs. These studies show that adaptive routing approaches exhibit a variety of requirements in crossbar size, routing arbitration complexity, and the number of virtual channels required to achieve deadlock-free adaptive routing. Using a dimension-order router as the baseline, cost estimates show that both planar-adaptive routing and the turn model can be implemented with modest increases in hardware. However, both produce a significant increase in setup delay and flow control latency. Applying our model to the *-channels algorithm shows that *-channels has slightly greater cost than the other adaptive routers, and differs in that it requires remote buffer status information to make routing decisions.

## 5.1 Dimension-order Router

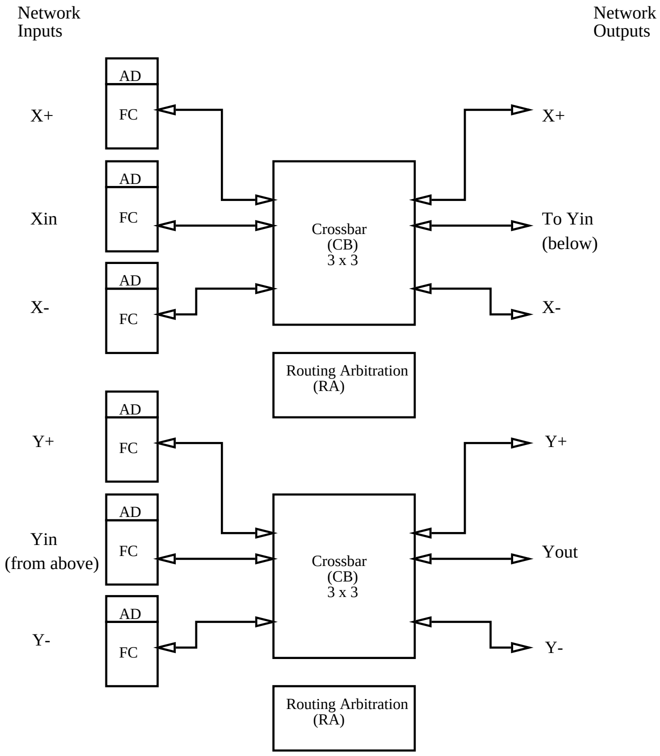


**Figure 5**: The block diagram for a dimension-order router. The 3x3 crossbars are networks of 3 input selectors.

Dimension order routing routes a packet successively in each dimension, until the distance in that dimension is zero, then proceeds to the next dimension. Numerous implementations of dimension-order routers have been constructed by both researchers and companies [14, 15, 19, 32, 9, 18, 31]. Applying our canonical router architecture produces a dimension-order router (see Figure 5) which corresponds to the

[2]Numbers presented are based on Mitsubishi Electronics M6007x and M6008x Series 0.8 micron gate array family.

basic organization of all of these designs.

The setup delay in a dimension-order router involves decoding the address (and updating the header), a trivial routing decision whether to continue in the current dimension or proceed to the next, connecting the crossbar, and sending data through the crossbar. Because dimension order routing only changes dimensions in order, there is one crossbar for each dimension, and only three inputs D+, D- and from a higher dimension. The three outputs are D+, D- and to a lower dimension. This highly specialized datapath partitions the router, allowing each part to run extremely fast. The routing decision involves only a check for zero and arbitration for the switch output. The equation for setup delay can be written as shown:

$$T_{DimOrder} = T_{AD} + T_{ARB} + T_{CB}$$

which based on our parameterized model and the fact that the dimension order router requires P = 3, F = 3, and V = 0, gives

$$T_{DimOrder} = c_3 + c_4 + c_5 * \log F + c_0 + c_1 * \log P$$

$$T_{DimOrder} = c_3 + c_4 + c_5 * \log 3 + c_0 + c_1 * \log P$$

which for the 0.8 micron CMOS gate array, gives a setup delay of 5.6 nanoseconds. The address update time can be overlapped with the later steps, but the address decoding time is on the critical path. Of this setup delay, 2.7ns are spent in the address decoder, 1.55ns are spent for the extremely simple arbitration, and 1.35ns are spent crossing the crossbar. The major contributor to the delay is the address decoding; other aspects of the path setup are comparatively inexpensive. Dimension-order routers do not require virtual channel controllers, so their large setup delay is not a factor.

Performing a flow control operation in a dimension-order router requires delay through the crossbar and the flow controller. The flow control delay can be written as shown:

$$T_{fc-DimOrder} = T_{FC} + T_{CB}$$

$$T_{fc-DimOrder} = c_2 + c_0 + c_1 * \log P$$

Substituting P = 3 gives 3.55 nanoseconds for a flow control operation in a dimension order router, an extremely low latency. This means that the router could achieve $280 \times 10^6$ flow control operations per second or $280 \times 10^6$ flits per second along any particular channel. The setup time and router cycle time compare favorably with a mature series of routers designed by Seitz's group at Caltech [31].[3]

As our studies will show, the dimension-order router is significantly simpler in both setup and flow control than any of the other adaptive routers, giving faster operation. However, this advantage is tenuous, as adding even a few virtual lanes (and the required virtual channel controllers) to dimension order routing can effectively eliminate the speed advantage [5, 4].

## 5.2 Planar-Adaptive Router

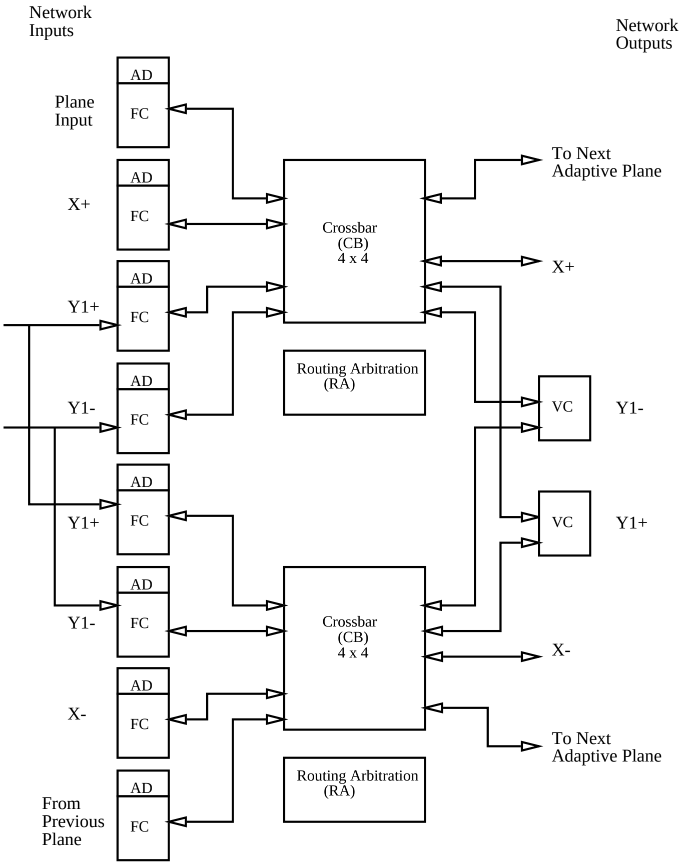


**Figure 6**: Block diagram for a planar-adaptive router. A slice for one adaptive plane is shown.

Planar-adaptive routing (PAR) has been proposed as an inexpensive form of adaptive routing [8]. PAR is a minimal adaptive router which reduces the cost of adaptive routing by limiting it to only two dimensions at a time. This approach limits the crossbar size to

[3]The point is not to claim that our design is better, but rather that we are not skewing the comparison against deterministic routers.

$4 \times 4$, routing decisions to only four alternatives, and the number of virtual channels to only three per physical channel. These requirements are independent of the number of dimensions in the network. The block diagram for a planar-adaptive router is shown in Figure 6. In contrast to the dimension-order router, the planar-adaptive router uses a partially specialized datapath for each adaptive plane.

Adaptive routers make routing decisions based on router state. Close examination of a router implementation reveals that this flexibility must incur some overhead. In particular, the routing decision logic becomes more complicated. First, the routing decision logic uses both the incoming packets and the current router state to make output assignments. Second, the address decoders must first decode the packet header and then update it based on the routing decision. *This update is particularly important, as it appears on the critical path for setup, and is present in all of the adaptive routers considered.* Our adaptive router designs each compute all possible new headers (the number depends on the routing freedom), then select the appropriate one. Using relative addresses allows completion of routing in a dimension to be a simple zero-check operation. Planar adaptive routers also require virtual channels for deadlock prevention, so the critical path consists of address decoding, routing arbitration, header selection, crossbar delay, and virtual channel controller delay. Thus the router setup delay for the PAR is:

$$T_{PAR} = T_{AD} + T_{ARB} + T_{SEL} + T_{CB} + T_{VC}$$

Plugging in the expressions from Figure 2,

$$\begin{aligned} T_{PAR} &= c_3 + c_4 + c_5 * \log F + c_6 + c_7 * \log F \\ &\quad + c_0 + c_1 * \log P + c_8 + c_9 * \log V \end{aligned}$$

using P = 4, F = 4, and V = 3 based on the deadlock-freedom requirements gives:

$$T_{PAR} = c_3+c_4+c_5*2+c_6+c_7*2+c_0+c_1*2+c_8+c_9*\log 3$$

which for our 0.8 micron CMOS gate array model gives a setup latency of 10.9 nanoseconds. This delay is nearly two times larger than that of the dimension-order router. Clearly, even inexpensive adaptive routing significantly increase router setup latency. Most of the increase is due to two things: the use of virtual channel controllers and header selection based on the routing decision. Both of these operations lie on the critical path, and hence slow the setup of connections through the router. Though some adaptive routers do not require virtual channels, all adaptive routers that use relative addressing require header selection on the critical path.

The flow control path in the planar-adaptive router is similar to that of the dimension-order router with one exception. Because virtual channels are required for deadlock prevention, the virtual channel controller must be part of the flow control path. The flow control latency in the planar-adaptive router can be written as follows:

$$T_{fc-PAR} = T_{FC} + T_{CB} + T_{VC}$$

$$T_{fc-PAR} = c_2 + c_0 + c_1 * \log P + c_8 + c_9 * \log V$$

Plugging in the values P = 4 and V = 3 for the PAR gives

$$T_{fc-PAR} = c_2 + c_0 + c_1 * 2 + c_8 + c_9 * \log 3$$

which based on our instantiated model gives a flow control delay of 6.15 nanoseconds.[4] This is almost two times that of the dimension-order router and a direct result of the introduction of virtual channel controllers into the critical path. Adding virtual channels incurs significant overhead and may directly affect the speed of signalling that a router can achieve.

## 5.3 The Turn Model

The turn model is an interesting alternative approach to adaptive routing which prevents deadlock without virtual channels [28]. The turn model encompasses a large collection of routing algorithms. However, there are a range of similar algorithms; we consider the *negative-first* algorithm. It can be used in arbitrary dimensioned networks, and is provably deadlock free. The idea is first to route adaptively in all of the negative directions, until there are no more, then route adaptively in all of the positive directions. Implementing a negative first routing algorithm basically requires crossbars from all inputs to all outputs (only a few connections are prohibited). In addition, arbitration for all inputs to all outputs is also necessary. Since deadlock freedom is assured by preventing turns, no virtual channels are required. A block diagram of a turn model router is shown in Figure 7.

The setup path for a turn model router is more complex than a dimension-order router's, but in a way that differs from the planar-adaptive router's. The turn model requires larger crossbars and larger arbitration

[4] However, a two-dimensional PAR only requires two virtual channels, reducing the setup delay and flow control times to 10.8ns and 5.8ns respectively.

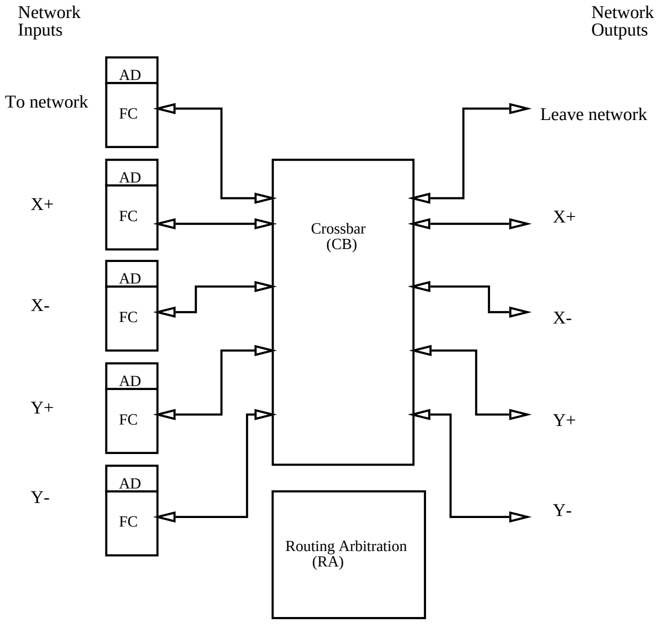


Figure 7: A block diagram for a Turn model router.

circuits, but no virtual channel controllers. Thus, the setup delay involves the address decoder, the router decision logic, the header selection logic, and finally the crossbar delay. Our model for router decision logic is probably conservative for a turn model router, which requires information across a number of dimensions in order to generate a set of acceptable outputs. The other routers (dimension order, planar adaptive, and *-channels) each generate such requests by considering only a few bits of the header independently. The setup delay for a two-dimensional turn-model router can be written as follows:

$$T_{Turn} = T_{AD} + T_{ARB} + T_{SEL} + T_{CB}$$

$$T_{Turn} = c_3 + c_4 + c_5 * \log F + c_6 + c_7 * \log F + c_0 + c_1 * \log P$$

where in this case, F and P are equal to two times the number of dimensions plus one. The number of dimensions is doubled because the routing algorithm does not allow the crossbar to be partitioned. The plus one is due to the network output port which delivers messages at their destination. Since the routing speed depends on the number of dimensions, in Figure 9, we give the setup delays for range of network dimensions. The setup delay can be written as below, where D is the network dimension.

$$\begin{aligned} T_{Turn-D} &= c_3 + c_4 + c_5 * \log(2D+1) + c_6 \\ &\quad + c_7 * \log(2D+1) + c_0 + c_1 * \log(2D+1) \end{aligned}$$

The Turn model router has a larger setup delay than the dimension-order router, due to the increased crossbar sizes and arbitration delay. For two dimensions, the setup time is 9.1ns. However, it is still less than the setup delay for the planar adaptive router, mostly because no virtual channels are required. For higher dimensions, the setup delay of the turn model router becomes larger due to crossbar size, routing arbitration, and header selection. These terms all grow with dimension, surpassing the cost of virtual channels in the planar-adaptive router at four dimensions.

The flow control delay for the Turn Model router has similar structure to that of the dimension order router, though the crossbar is a little larger. It can be written as follows:

$$T_{fc-Turn} = T_{FC} + T_{CB}$$

Which plugging in our parameterized model gives

$$T_{fc-Turn} = c_2 + c_0 + c_1 * \log P$$

As before, the speed depends on the dimension of network, so with D as the network dimension:

$$T_{Dfc-Turn} = c_2 + c_0 + c_1 * \log(2D+1)$$

A range of flow control cycle delays can be found in Figure 9.

The flow control delay is also affected by the size of the crossbar, producing a larger flow control delay than in the dimension-order router. However, the flow control delay of a turn model router is much lower than that in the planar-adaptive router because no virtual channel controllers are required. The two dimensional turn model router, has a flow control delay of 4.0 nanoseconds, only 13% larger than that for a dimension-order router. In three and four dimensional networks, the flow control delay is only 21% and 27% larger respectively. The turn model router's flow control delay is smaller than that of the planar-adaptive router even beyond ten dimensions, due to the absence of virtual channel controllers. Avoiding the requirement of virtual channel controllers simplifies routers, giving significant performance benefits. A more detailed analysis of these issues can be found in [5, 4].

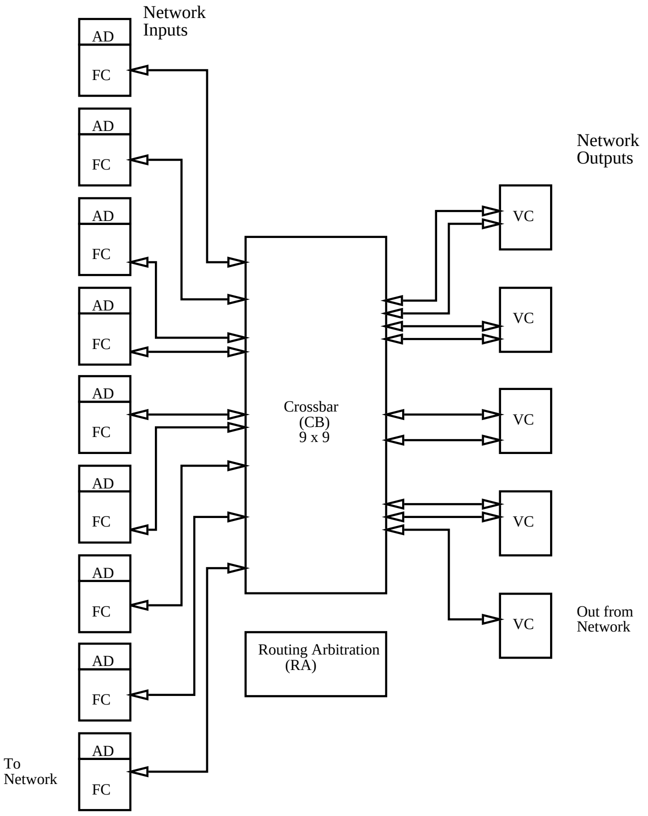


**Figure 8**: The architecture of a two-dimensional *-channels router. It requires both large crossbars and virtual channel controllers.

## 5.4 *-Channels Router

The *-channels router is interesting to compare to the other routers because it provides deadlock-free, fully-adaptive minimal routing with a modest number of virtual channels [6]. *-channels prevents deadlock in the fashion described by Duato's theory [17]. *-channels differs from the planar-adaptive router which provides only limited adaptivity and the turn model which is a non-minimal router. The *-channels router uses virtual channels to separate the adaptive routing channels from those reserved for dimension-order routing.

The *-channels router prevents deadlock by dynamically disallowing paths based on occupancy of buffers in neighboring nodes. This means that routing decisions depend critically on the availability of accurate buffer status from neighboring nodes. This issue is discussed below. For now, we assume that buffer status information is available. Casting a *-channels router into our canonical architecture produces a router as shown in Figure 8. The crossbar is $(4D+1)\times(4D+1)$, or $9 \times 9$ for the two-dimensional case shown. There is a separate crossbar port and flow control unit for each virtual channel. And, there is one VC for each physical channel.

The setup latency for the best case, assuming no stalling for a previous packet, can be written as follows:

$$T_{SC} = T_{AD} + T_{ARB} + T_{SEL} + T_{CB} + T_{VC}$$

Plugging in the expressions from Figure 3,

$$\begin{aligned} T_{SC} &= c_3 + c_4 + c_5 * \log F + c_6 + c_7 * \log F + c_0 \\ &\quad + c_1 * \log P + c_8 + c_9 * \log V \end{aligned}$$

For a two-dimensional network router, P = 9, F = 9, and V = 2 gives:

$$T_{SC} = c_3 + c_4 + c_5 * \log 9 + c_6 + c_7 \log 9 + c_0 + c_1 * \log 9 + c_8 + c_9$$

The *-channels routers includes the most costly alternatives, requiring a large crossbar switch, virtual channels, and maximal routing freedom, all of which combine to increase router latency. In return, the primary advantage of the *-channels router is that it provides fully-adaptive minimal routing. The setup delay of the *-channels router for two-dimensional networks is 12.65ns, poorer than all three of the other routers considered. Setup delays for a number of dimensions are shown in Figure 9. The setup delay for *-channels is longer because it requires the worst implementation features of both the planar adaptive router and the turn model router: virtual channels AND large crossbar switches. Furthermore, the *-channels router's performance disadvantage gets worse for higher dimensions, extracting a significant cost for allowing all minimal paths.

Thus, even in the best case, its latency is significantly worse than any of the other routers we have considered. If the networks are run close to capacity, and short messages are common, the need for accurate buffer status information from neighboring nodes could further increase setup delay. Because the node latency depends on whether one packet follows another directly, the maximum incurred latency for one packet following another can be as much as $2T_f$ where $T_f$ is the time of flight between routers. If a router implementation includes significant FIFO buffering the delay may increase further. This delay arises from the need to assure that the preceding packet has been moved out of the FIFO and across the crossbar. If

| | Network Dimension (n) | | | | |
|---|---|---|---|---|---|
| Routing Algorithm | 2 | 3 | 4 | 5 | 10 |
| Dimension order | 3,348 | 5,022 | 6,696 | 8,370 | 16,740 |
| Planar adaptive | 6,344 | 9,516 | 12,688 | 15,860 | 31,720 |
| Turn Model | 3,250 | 5,194 | 7,506 | 10,186 | 29,106 |
| *-Channels | 8,766 | 14,998 | 22,702 | 31,878 | 99,838 |

Figure 10: Gate counts for adaptive routers, based on the module gate count models.

it had not, then it would be unsafe to route a packet along that path.[5] This delay increases node latency and reduces channel bandwidth. Such delays will become more significant as clock rates and the internode propagation delay in cycles the available silicon for buffering increase.

The flow control latency in the *-channels router is nearly identical to that in the planar-adaptive router, requiring the involvement of the flow controller, a slightly larger crossbar, and a virtual channel controller.

$$T_{fc-SC} = T_{FC} + T_{CB} + T_{VC}$$

$$T_{fc-SC} = c_2 + c_0 + c_1 * \log P + c_8 + c_9 * \log V$$

which for two dimensions, the values P = 4D + 1 = 9 and V = 2 gives

$$T_{fc-SC} = c_2 + c_0 + c_1 * \log 9 + c_8 + c_9 * \log 2$$

which based on our instantiated model gives a flow control delay of 6.5 nanoseconds. This is larger than all of the other routers: dimension-order router, turn model, and planar-adaptive routers. The increased flow control latency is directly attributable to the requirement for a large crossbar, arbitration across the entire switch, and virtual channel controllers along the critical path. As with the setup delay, this disadvantage increases with respect to the dimension-order and planar adaptive routers as the dimension is increased.

## 5.5 Router Gate Counts

Router gate counts, based on the module gate count formulas in Figure 4 and the number of modules each routing algorithm requires, are shown in Figure 10. These estimates are conservative as they do not include gates required for input/output buffering, pads, and synchronization. However, they do show that hardware complexity of adaptive routers is manageable with current technology. This is particularly true for the low-dimensional networks which are the most attractive candidates.

[5] For more details see [6].

| Router | n | Setup Delay | FC Cycle |
|---|---|---|---|
| Dimension order | any | 5.6 ns | 3.55 ns |
| Planar adaptive | any | 10.9 ns | 6.15 ns |
| Turn Model | 2 | 9.1 ns | 4.0 ns |
| | 3 | 10.0 ns | 4.3 ns |
| | 4 | 10.6 ns | 4.5 ns |
| | 5 | 11.2 ns | 4.7 ns |
| | 10 | 12.8 ns | 5.1 ns |
| *-Channels | 2 | 12.7 ns | 6.5 ns |
| | 3 | 13.6 ns | 6.8 ns |
| | 4 | 14.3 ns | 7.1 ns |
| | 5 | 14.8 ns | 7.2 ns |
| | 10 | 16.6 ns | 7.4 ns |

**Figure 9**: Setup delays and flow control cycle times for a variety of adaptive routers. n is the network dimension; FC cycle is the flow control cycle time. All times in nanoseconds.

## 5.6 Other Routing Alternatives

There are many other well-known adaptive, wormhole routing approaches, but we could not consider all of them in the space of one paper. Several were excluded because they appear to be significantly more complex than the routers included. In particular, the Dally-Aoki router [12], the Linder-Harden router [24], the Ngai-Seitz router [27] fall into this category. Another interesting approach, the Chaos router [22], requires a significantly different basic router architecture based on virtual cut-through routing, and therefore was omitted. Basically, we tried to choose an interesting variety of the simplest adaptive, wormhole routing algorithms. Even they required significant hardware and additional latency beyond that required for a dimension-order router. It is our expectation that the more complex algorithms will suffer even more severe penalties due to larger requirements in terms of complex routing logic and virtual channels.

### 5.7 Discussion and Analysis

The purpose of developing a parametric speed model is to compare the cost and complexity of a variety of routing algorithms. In the process, the detailed analysis required to design such routers gives insight into the critical paths and decisions in a router design. Based on our experience, we can make a number of observations on how a routing algorithm affects the complexity and speed of its implementation.

- All adaptive routers require a serialized routing decision and header update which increases their setup delay relative to dimension order routers.
- Even computing all possible new headers in parallel does not eliminate this cost, as even selecting the appropriate modified header incurs latency. Absolute network addressing can help this problem, but will incur increased routing decision cost (longer compare times).
- Virtual channels are expensive in terms of setup latency and flow control cycle time.
- Crossbar growth effects are small[6], though our cost model does not include the physical layout and interconnection issues.
- Arbitration costs for increased routing freedom are as significant as crossbar growth effects.
- Adaptive routers should not claim advantages in low-load latency as any apparent advantage is probably eliminated by increases in router setup time.
- Some adaptive routers may have a significant potential throughput disadvantage, based on the flow control cycle time.

Though most routers have employed relative addressing, its cost is significant in adaptive routers, and fixed addressing may have to be reconsidered to reduce header selection cost. Virtual channels are quite expensive, and requiring them appears to be a significant performance disadvantage. However, their high cost means that deciding to include them in a deterministic router to support virtual lanes might well narrow or effectively eliminate the speed gap between deterministic and some adaptive routers. Surprisingly, the effect of requiring larger crossbars is modest; router arbitration contributes nearly an equal portion to delay.

[6] Due in part to limiting ourselves to modest dimension and numbers of virtual lanes.

With respect to the large number of adaptive routing simulations normalized on router cycle time and assuming constant latency per node, it is clear that the setup latency of adaptive routers is likely to be significantly larger. This means that apparent advantages in low load latency in router cycles are probably illusory. Further, simulations demonstrating higher channel utilization assuming the same flow control rate are also probably not realistic. If the slower flow control speeds of adaptive routers translate into lower speed signalling, even throughput advantages of adaptive routers on nonuniform loads may be erased by the brute speed advantage of the deterministic routers. To be competitive, adaptive routers will have to include higher speed signalling than the flow control rate, using larger flits and splitting each flit into a number of phits, to provide a real throughput advantage.

### 5.8 Internode Delays

In this paper, we have focused exclusively on intra-router delays. In complete systems, the time of flight from one router chip to another also contributes to the effective routing delay. If inter-router delay is significant, it reduces the importance of differences in intra-router delay. A variety of studies show chip crossing times from below one to as many as several nanoseconds [20, 25] with the primary determining factors being the electrical characteristics of the wires, determined by packaging choices. If the system is asynchronous, or there is significant clock skew, additional margins are required to acquire synchronization at these chip crossings. As a result, the chip crossing and synchronization time can be significant, reducing the impact of differences in setup delay and flow control cycle time.

## 6 Related Work

A number of other research projects have explored router implementation issues. Some of the earliest involve techniques for deadlock-free wormhole routing [14] and were based on full crossbar architectures. Later studies optimized the internal architectures based on the regularity of dimension-order routing, partitioning crossbars and subdividing the router into a set of smaller, identical router automata. These organizational improvements produce significant performance improvements, culminating in an organization similar to the router architecture shown in Figure 5 [15, 19, 31]. However, these studies have focused on dimension-order routing and have not directly considered the cost of adaptive routing. A number of commercial wormhole routers have also been constructed for the Symult 2010 [32] and the Intel

Paragon [9, 18]. The Paragon router is of particular interest as it is implemented in a similar technology (0.8 micron CMOS gate array) and gives performance comparable to our router designs. Public figures for its delay and channel bandwidth are 40 nanoseconds and 200 megabytes/second respectively.

## 7 Summary and Conclusion

We have developed a parametric cost model for wormhole routers. This model captures the cost and achievable speed of the basic routing functions, providing a basis for calibrating simulation studies and the complexity of a wide variety of routing algorithms. Instantiating this for a particular technology, CMOS gate arrays, based on a family of router designs [4], provides one set of constants for the parameterized model, allowing the direct comparison and evaluation of several specific routing algorithms on the basis of router setup delay and flow control speed.

Applying the instantiated model to several deterministic and adaptive routing algorithms shows that adaptive routers are generally significantly more complex. In particular, they require virtual channels (and therefore the virtual channel controllers that incur significant delay) or larger crossbars and routing decision circuitry. The net effect is to produce slower routers. Of particular interest is the fact that all adaptive routers suffered from increased delay due to waiting for a routing decision and then selecting an appropriate updated header. This serialization is much less in deterministic routers, so it incurs a significant relative increase in setup delay for adaptive routers. A number of other specific insights as to the relative costs of different adaptive routing algorithms were also presented.

- Header update and selection is expensive in adaptive routers; absolute addressing should be reconsidered.
- Virtual channels are expensive in terms of latency and cycle time, so decisions to include them to support adaptivity or even virtual lanes should not be taken lightly.
- Larger crossbars and more complex arbitration cause some increase in the complexity of adaptive routers, but the rate of increase is small.

However the broadest insight is that the complexity of adaptive routers significantly increases their setup delay and flow control cycle times, implying that claims of performance advantages in channel utilization and low load latency must be carefully balanced against losses in achievable implementation speeds.

Beyond illuminating the difference between adaptive routing algorithms, the objective of this work is to provide a cost and speed model which can be used to calibrate and fairly evaluate simulation results on the performance of adaptive routing. In effect, this can provide a level playing field where the benefits of adaptivity can be intelligently evaluated and traded off against its cost. In this vein, this paper is intended as neither a critique nor a promotion of adaptive routing but rather to catalyze adaptive routing researchers to substantiate their claims in a more convincing fashion.

## Acknowledgements

The research described in this paper was supported in part by National Science Foundation grants CCR-9209336 and MIP-92-23732, Office of Naval Research grant N00014-92-J-1961, and National Aeronautics and Space Administration grant NAG 1-613. Additional support has been provided by a generous special-purpose grant from the AT&T Foundation.

The paper has benefited greatly from careful proofreadings by Jae Kim, Scott Pakin, and Harpinder Madan. The parametric model presented in this paper is based on the router designs done by Kazuhiro Aoyama. The studies would not have been possible without the cooperation of Mitsubishi Electric of America; particularly that of Mr. Kenji Baba and Mr. Anil Chaudhry.

## References

[1] A. Agarwal et. al. The mit alewife machine: A large-scale distributed-memory multiprocessor. Lcs-technical memo 454, Massachusetts Institute of Technology, Laboratory for Computer Science, 1991.

[2] A. Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, 1991.

[3] G. Alverson, R. Alverson, D. Callahan, B. Koblenz, A. Porterfield, and B. Smith. Exploiting heterogeneous parallelism on a multithreaded multiprocessor. In *Proceedings of the 6th ACM Interational Conference on Supercomputing*, 1992.

[4] K. Aoyama. Design issues in implementing an adaptive router. Master's thesis, University of Illinois, Department of Computer Science, 1304 W. Springfield Avenue, Urbana, Illinois., January 1993.

[5] K. Aoyama and A. A. Chien. The cost of adaptivity and virtual lanes in a wormhole router. *submitted for publication*, 1993.

[6] P. Berman, L. Gravano, G. Pifarre, and J. Sanz. Adaptive deadlock and livelock free routing with all minimal paths in torus networks. In *Proceedings of the Symposium on Parallel Algorithms and Architectures*, 1992.

[7] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski, and J. Webb. iwarp: An integrated solution to high-speed parallel computing. In *Proceedings of Supercomputing '88*, pages 330–341. IEEE Press, 1988. Orlando, Florida.

[8] A. A. Chien and J. H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. In *Proceedings of the International Symposium on Computer Architecture*, pages 268–77, May 1992.

[9] Intel Corporation. Paragon XP/S product overview. Product Overview, 1991.

[10] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5), 1987.

[11] W. J. Dally, S. Ahmed, P. Carrick, A. Chien, R. Davison, J. Fiske, G. Fyler, W. Horwat, J. Keen, S. Lear, R. Lethin, M. Vestrich, T. Nguyen, M. Noakes, P. Nuth, and D. Wills. Design and implementation of the message-driven processor. In *Proceedings of the 1992 Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems, T. Knight and J. Savage, eds.*, pages 5–25. MIT Press, 1992.

[12] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks usin g virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, To appear.

[13] W. J. Dally, J. A. S. Fiske, J. S. Keen, R. A. Lethin, M. D. Noakes, P. R. Nuth, R. E. Davison, and G. A. Fyler. The message-driven processor. *IEEE Micro*, April 1992.

[14] W. J. Dally and C. Seitz. The torus routing chip. *Distributed Computing*, pages 187–96, 1986.

[15] W. J. Dally and P. Song. Design of a self-timed vlsi multicomputer communication controller. In *Proceedings of the International Conference on Computer Design*, pages 230–4. IEEE Computer Society, 1987.

[16] J. Draper and J. Ghosh. Multipath e-cube algorithms (meca) for adaptive wormhole routing and broadcasting in k-ary n-cubes. In *The Fifth International Parallel Processing Symposium*, 1992.

[17] J. Duato. On the design of deadlock-free adaptive routing algorithms for multicomputers: design methodologies. In *Proceedings of Parallel Architectures and Languages Europe*, 1991.

[18] Dave Dunning. The intel paragon router. VLSI Seminar Talk at the Massachusetts Institute of Technology, 1992.

[19] Charles M. Flaig. Vlsi mesh routing systems. Master's thesis, California Institute of Technology, Department of Computer Science, May 1987.

[20] R. Kaw. Comparison of chip crossing delay in various packaging environments. In *Proceedings of the International Conference on Computer Design*, pages 233–6. IEEE Computer Society Press, 1989.

[21] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communications switching technique. *Computer Networks*, 3(4):267–86, 1979.

[22] S. Konstantinidou and L. Snyder. Chaos router: Architecture and performance. In *Proceedings of the International Symposium on Computer Architecture*, pages 212–21, 1991.

[23] D. Lenoski, J. Laudon, K. Gharacharloo, W. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. Lam. The stanford dash multiprocessor. *IEEE Computer*, March 1992.

[24] D. Linder and J. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers*, C-40(1):2–12, January 1991.

[25] Mitsubishi Electronic Devices Group. *Data Book for the Mitsubishi M6007x and M6008x Series Gate Array Families*, November 1992.

[26] NCUBE, Beaverton, Oregon. *NCUBE 2 6400 Series Supercomputer: Technical Overview*, 1989.

[27] J. Ngai and C. Seitz. A framework for adaptive routing in multicomputer networks. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, 1989.

[28] L. Ni and C. Glass. The turn model for adaptive routing. In *Proceedings of the International Symposium on Computer Architecture*, 1992.

[29] P. Nuth and W. Dally. The j-machine network. In *Proceedings of the 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 420–23, 1992.

[30] C. Peterson, J. Sutton, and P. Wiley. iwarp: a 100-mops liw microprocessor for multicomputers. *IEEE Micro*, June 1991.

[31] C. Seitz and W. Su. A family of routing and communication chips based on the mosaic. In *Proceedings of the University of Washington Symposium on Integrated Systems*, 1993.

[32] C. L. Seitz, W. C. Athas, C. M. Flaig, A. J. Martin, J. Seizovic, C. S. Steele, and W. Su. The architecture and programming of the ametek series 2010 multicomputer. In *Proceedings of the Third Conference on Hypercube Computers*, pages 33–6. Association for Computing Machinery, ACM Press, January 1988.