A Delay Model for Router Microarchitectures

GIVEN ROUTER PARAMETERS, THIS DELAY MODEL PRESCRIBES REALISTIC PIPELINES, ENABLING ROUTER ARCHITECTS TO OPTIMIZE NETWORK PERFORMANCE BEFORE BEGINNING ACTUAL DETAILED DESIGN.

Li-Shiuan Peh
William J. Dally
Stanford University

Buses and crossbars have traditionally served as the communication fabrics of computer systems, network switches and routers, and other digital systems. However, demand for communication bandwidth is overwhelming the limits of these fabrics, and interconnection networks, historically used in high-end multiprocessor systems,[1-3] have surfaced as an alternative. In recent years, interconnection network fabrics have been deployed in high-speed network switches[4] and high-bandwidth core Internet routers.[5]

These interconnection networks' performance critically depends on the performance of the routers from which they are constructed. Armed with accurate router performance models, interconnection network architects can optimize and fine-tune router parameters before starting detailed design.

This article introduces a router delay model that takes into account the pipelined nature of contemporary routers and proposes pipelines matched to the specific flow control method employed. Given the type of flow control (see the "What is flow control?" sidebar) and router parameters, the model returns router latency in technology-independent units and the number of pipeline stages as a function of cycle time.

We apply this model to derive realistic pipelines for wormhole and virtual-channel routers and compare their performance. Contrary to the conclusions of previous models, our results show that the latency of a virtual-channel router doesn't increase as we scale the number of virtual channels up to 8 per physical channel. Our simulation results also show that a virtual-channel router gains throughput of up to 40% over a wormhole router.

Compared with the commonly assumed unit-latency model, which ignores implementation complexity and assumes router latency to be a single cycle, our model shows significant performance differences: 56% in zero-load latency and 30% in throughput. This highlights the importance of considering implementation costs when simulating router performance.

Previous models

Chien[6] first noted the need for router delay models that consider implementation complexity and proposed a model for wormhole and virtual-channel routers. Chien's model uses the router architecture shown in Figure 1, which was employed in the Torus Routing Chip,[7] for all routers regardless of the flow control method. It defines per-hop router latency as the total delay of the functions on the critical path. These functions are address decoding (AD), routing and crossbar arbitration (RA), crossbar traversal, and virtual-channel allocation (VC). Through detailed gate-level design and analysis, Chien express-

0272-1732/01/$10.00 © 2001 IEEE

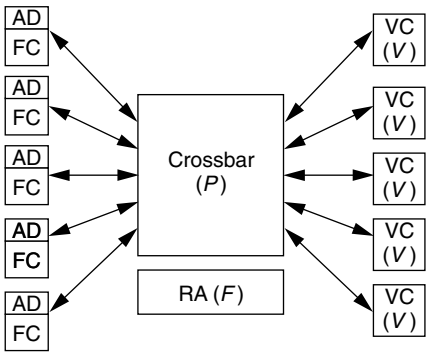

Figure 1. Canonical router architecture proposed in Chien's model. The delay model's parameters are *P*, the number of ports on the crossbar; *F*, the number of output route choices; and *V*, the number of virtual channels per physical channel.

es these functions' delays in parameterized equations grounded in a 0.8-micron CMOS process. By substituting the parameters of a given router into Chien's equations, a designer can obtain estimates of router latency in the 0.8-micron process.

However, although most contemporary routers are heavily pipelined, Chien's model doesn't account for pipelining, assuming instead that the entire critical path fits within a single clock cycle. Therefore, Duato and Lopez[8] proposed an extension of Chien's model to pipelined routers. Their model prescribes a fixed three-stage pipeline. The routing stage contains the address decoding, routing, and arbitration functions of Chien's model; the switching stage includes crossbar traversal; and the channel stage includes virtual-channel allocation and internode delay.

But these models have significant shortcomings. First, they both assume that clock cycle time depends solely on router latency. In practice, routers are heavily pipelined, making cycle time largely independent of router latency. Typically, router designers must work within the limits of a clock cycle determined by factors beyond the router, such as the fundamental limits of chip-to-chip signaling or the processor clock cycle. A realistic delay model must work in an environment where the cycle time is fixed and the number of pipeline stages is variable.

What is flow control?

In an interconnection network, routing policy determines the path a packet takes from source to destination, while flow control policy determines how the packet moves along the path. By governing when packets receive buffers and channels, flow control policy significantly shapes an interconnection network's performance. A poor flow control policy can result in a network that saturates at 30% capacity, whereas a good flow control policy enables a network to operate at 80% or higher capacity.

Dally and Seitz first introduced wormhole flow control to enable fast single-chip routers to be built in 1986 technology.[1] It improves upon prior flow control policies by allocating buffers and channels to flits instead of packets. A flit, which is a part of a packet, can leave for the next hop once buffering is available to hold the flit. For the same amount of storage, this results in lower latency and greater throughput, since a flit need not wait for enough buffering for the entire packet. However, wormhole flow control uses channels inefficiently. Although buffers are allocated in flit-size units, a physical channel is held for the duration of a packet. As a result, when a packet is blocked, all physical channels held by this packet are left idle, and other packets queued behind the blocked packet are unable to use the idle channels.

Virtual-channel flow control alleviates this problem by associating several virtual channels, each with a separate flit buffer queue, with a physical channel.[2] Thus, when a packet blocks while holding a virtual channel, other packets can still traverse the physical channel through other virtual channels, leading to higher throughput.

The Myrinet high-speed switch[3] is an example of a straight wormhole router; the Cray T3E router[4] and the Avici Terabit Switch/Router[5] are examples of virtual-channel routers.

References

1. W.J. Dally and C. Seitz, "The Torus Routing Chip," *Distributed Computing*, vol. 1, no. 3, 1986, pp. 187-196.
2. W.J. Dally, "Virtual-Channel Flow Control", IEEE *Trans. Parallel and Distributed Systems*, vol. 3, no. 2, Mar. 1992, pp. 194-205.
3. N.J. Boden et al., "Myrinet—A Gigabit-per-Second Local-Area Network," *IEEE Micro*, vol. 15, no. 1, Feb. 1995, pp. 29-36.
4. Cray Inc., http://www.cray.com/products/systems/crayt3e/paper1.html.
5. Avici Systems Inc., http://www.avici.com.

Second, these models also attempt to fit all routers into a single canonical architecture, resulting in a mismatch between the architecture and the flow control method. For instance, crossbar passage is arbitrated on a per-packet basis and held throughout a packet's duration. This prompts the need for a huge crossbar in a virtual-channel router, with the number of ports equal to the total number of virtual channels. The result is unnecessary delay in the crossbar arbitration and traversal functions. Buffering flits (parts of packets) at virtual-channel controllers, whose arbitration delay increases with the number of virtual channels, also adds needless cost to a virtual-channel router.

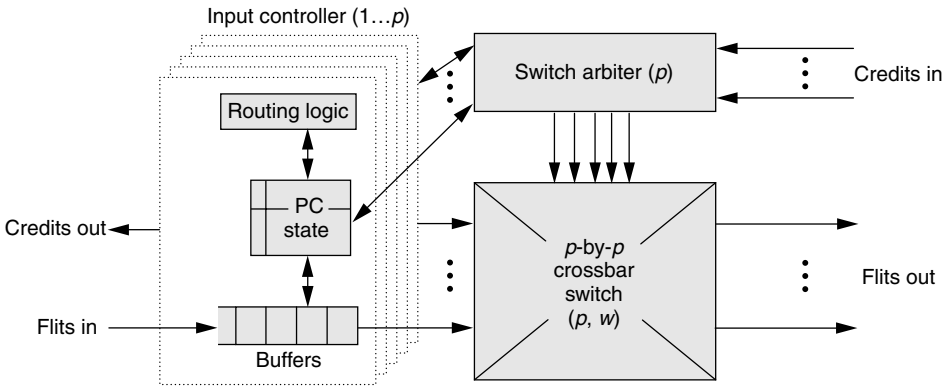

Figure 2. Canonical wormhole router architecture, with p physical channels, each w bits wide.

Proposed model

Our proposed router delay model consists of a general router model and a specific router model. The general model outlines a design methodology for pipelining a router given a clock cycle time, using delay estimates derived by the specific model.

General router model

To illustrate how the model works, we apply it here to wormhole and virtual-channel routers. We start with the definition of canonical router architectures suited for each flow control method.

Canonical router architectures. Figure 2 illustrates the canonical wormhole router architecture. The parameters affecting the delay of a wormhole router's various modules are p, the number of physical channels, and w, the channel width.

Consider a two-flit packet, consisting of one head flit and one tail flit, traversing the canonical wormhole router of Figure 2 from the injection input channel to the eastern output channel. The packet proceeds through the states of routing, switch arbitration, and switch traversal. When the head flit arrives, the input controller decodes its type field. Finding it a head flit, the input controller forwards the destination field to the routing logic, buffers the entire flit in the input queue, and sets the channel state (PC state) to routing.

The routing logic returns the output port for this packet—in this case, the eastern output port. At this point, the input controller sets the channel state to switch arbitration and sends a request for the eastern output port to the global switch arbiter. The global switch arbiter receives arbitration requests from all input controllers, resolves contention, and assigns available output ports to the requesters. In this case, the arbiter grants the eastern output port to the request from the injection input channel and marks this output port as unavailable to other requests.

Upon receipt of this grant from the global switch arbiter, the input controller sets the injection input channel's state to switch traversal. If a free buffer is available on the next hop to hold the flit, the input controller reads the packet's head flit from the input queue and sends it through the crossbar. The switch arbiter sets the crossbar configuration; in this example, it has connected the injection input port to the eastern output port. Thus, the head flit traverses the crossbar and proceeds over the output link for the next hop.

When the next flit arrives, the input controller decodes its type field and finds it to be a tail flit. Since it isn't a head flit, it needn't go through routing or switch arbitration. Instead, the input controller simply buffers it in the input queue and sends it to the output port reserved by the head flit. When the tail flit leaves the input queue, it releases the resources held by the packet by setting the channel state to idle and signaling the global switch arbiter to free the reserved output port.

Whenever a flit leaves, the number of buffers on the next hop is decremented, and a credit is returned to the previous hop, prompting its next-hop buffer count to be incremented.

Figure 3 shows the canonical router architecture for virtual-channel flow control. Its additional parameter is v, the number of virtual channels per physical channel. This architecture differs from that proposed in Chien's model, adopting a more efficient design in which crossbar ports are shared across the virtual channels of a physical channel and allocated flit by flit. Hence, instead of a crossbar with pv ports, our architecture has a crossbar with just p ports.

The virtual-channel router has a separate input queue and a separate copy of the channel state (VC state), for each virtual channel. When the packet's head flit arrives at the injection channel's input controller, the controller decodes the flit's virtual-channel identifier (VCID) field and buffers the entire flit in the appropriate flit queue.

Consider the same two-flit packet flowing through a virtual-channel router. The packet proceeds through routing, switch arbitration, and traversal as described earlier, with the addition of a virtual-channel allocation state. Assuming the head flit has VCID 0, the input controller then injects the packet into virtual channel 0 (VC 0) of the injection channel and buffers it accordingly into queue 0. Then, VC 0 enters the routing state and sends the flit's destination field to the routing logic, which returns the output virtual channels (not physical channels) that the packet can use.

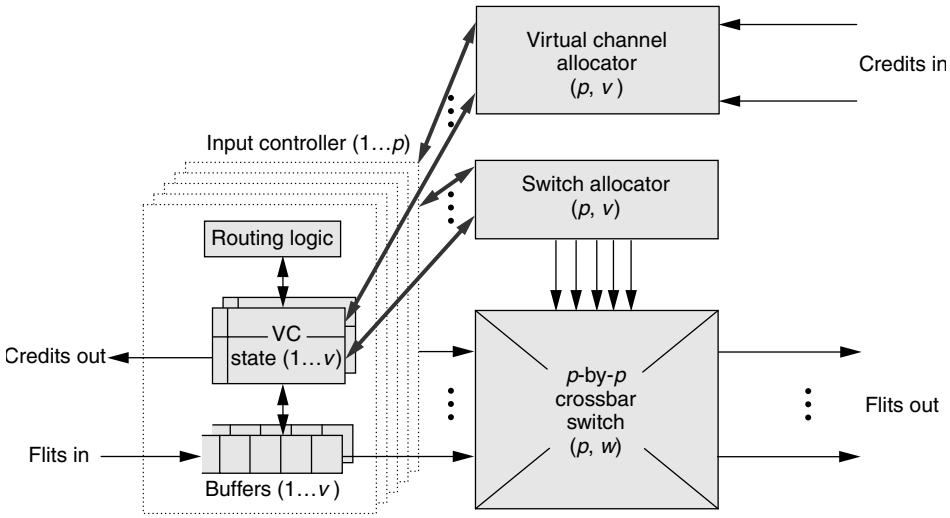

Figure 3. Canonical virtual-channel router architecture, with p physical channels, each w bits wide, and v virtual channels per physical channel. The boldface arrows represent v requests or responses.

When the input controller receives these output virtual channels from the routing logic, it sets VC 0's state to virtual-channel allocation. VC 0 then requests these output virtual channels from the global virtual-channel allocator. The global virtual-channel allocator collects all the requests from each input controller and returns available output virtual channels to successful requesters. It also updates the status of these output virtual channels as unavailable.

When VC 0 is allocated an output VC—say, output VC 1 of the eastern port—the head flit proceeds to the next step and sends requests for the eastern output port to the global switch allocator if the next-hop buffer count for VC 1 is not zero. Instead of reserving output ports for a packet's entire duration, the virtual-channel router's switch allocator allocates crossbar passage to flits of different packets on a cycle-by-cycle basis. Once this head flit secures passage to the eastern output port, it leaves for the crossbar and goes to the next hop, with its VCID field updated to VC 1.

Each time a flit leaves, the buffer count for its output VC—VC 1 in this example—is decremented. A credit containing the input VC—VC 0 in this case—is sent to the previous router, prompting it to increment its buffer count.

When the subsequent tail flit arrives, the input controller enqueues it according to its VCID field. It then inherits the output VC reserved by its head flit and submits a request to the global switch allocator for the eastern output port. Once granted crossbar passage, it signals the virtual-channel allocator to release the reserved output VC and leaves for the next hop, with its VCID field updated to VC 1. Note that in this canonical architecture, the function of multiplexing virtual channels onto a physical channel falls on the switch allocator, instead of the virtual-channel controllers of Chien's architecture.

Atomic modules and dependencies. The canonical router architectures in Figures 2 and 3 include modules that are not easily pipelined because they contain state that depends on the module output. These modules, which we call atomic modules in our model, are best kept intact within a single pipeline stage. An example of an atomic module is the virtual-channel allocator in a virtual-channel router. If this module straddles multiple pipeline stages,

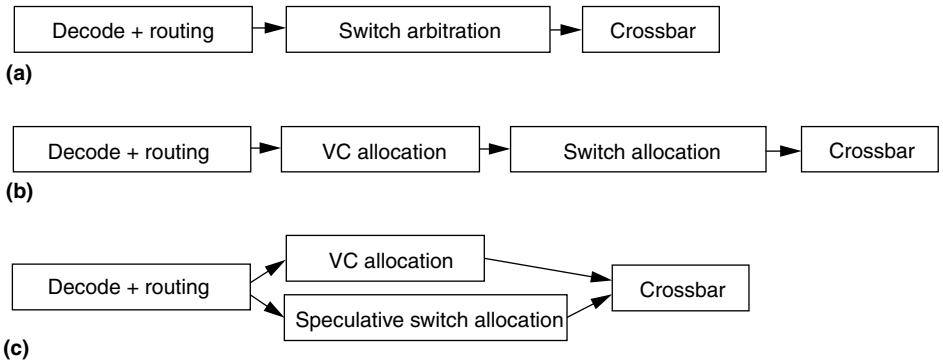

Figure 4. Atomic modules and dependencies of a wormhole router (a), a virtual-channel router (b), and a speculative virtual-channel router (c).

grants may not be reflected correctly before the next allocation. Although there are many ways to pipeline a virtual-channel allocator if it requires more than a single clock cycle, performance will usually suffer. Another module that should remain within a single pipeline stage is a separable allocator. This atomic module has a large number of wires connecting the input and output ports, which will require excessive latching if the allocator is partitioned across multiple pipeline stages.

Figure 4 shows the various atomic modules of wormhole and virtual-channel routers in our model. These atomic modules' inputs can depend on another module's outputs. Such dependencies determine a router's critical path. Figures 4a and 4b show the basic dependencies of wormhole and virtual-channel routers. (Because we are emphasizing a comparison of flow control techniques, we view routing as a black box and assume that decoding and routing take one clock cycle.)

Sometimes, speculation can avert these dependencies. For instance, the switch allocator in a virtual-channel router can speculatively assume that the packet will succeed in obtaining a free output virtual channel from the virtual-channel allocator. Thus, it can request the desired output port before it has secured an output virtual channel. Should the speculation be incorrect, the reserved crossbar passage will be wasted. With speculation, a virtual-channel router removes the dependency between the virtual-channel allocation and switch allocation modules and cuts down on its critical path delay, as Figure 4c shows. In this article, we model only basic wormhole and virtual-channel routers. We present the details of a speculative virtual-channel router in another publication,[9] where we show that a speculative virtual-channel router's latency can be reduced to that of a wormhole router.

Pipeline design. To arrive at a realistic pipeline design, we model the delay of each atomic module with parametric equations derived by the specific router model. The specific router model generates two delay estimates: latency (t_i) and overhead (h_i). Latency spans the time beginning when inputs are presented to the module and ending when the outputs needed by the next module are stable. Overhead is the delay expended by additional circuitry required before the next set of inputs can be presented to the module. In a switch arbiter, for instance, latency spans from the point when requests for switch ports are presented to the point when grant signals are stable. Overhead is the delay that occurs when the arbiter updates priorities between requesters in preparation for the next set of requests.

Armed with t_i and h_i of each atomic module on the critical path and the clock cycle time *clk*, the model prescribes the router's pipelining as follows: Given *a*, the first atomic module, and *b*, the last atomic module in the pipeline stage,

$$\sum_{i=a}^{b} t_i + h_b \le clk \quad \text{and}$$

$$\sum_{i=a}^{b+1} t_i + h_{b+1} > clk \quad \text{and} \quad \sum_{i=a-1}^{b} t_i + h_b > clk$$

Specific router model

We use the specific router model to derive the parameterized delay equations for atomic modules. We base our specific delay model on the theory of logical effort[10] proposed by Sproull and Sutherland for estimating circuit delay and guiding the design of minimum-delay circuits.

In the following equation, we calculate circuit delay T (in τ, the delay of an inverter with identical input capacitance[11]) along a path as

Table 1. Parameterized delay equations (in τ) for wormhole and virtual-channel routers ($1\tau_4 = 5\tau$).

Module	Parameterized delay equations (in τ)	Model (τ_4) ($p = 5$; $w = 32$; $v = 2$; clk = $20\tau_4$)	Synopsys timing analyzer (τ_4)
Wormhole router			
Switch arbiter (swarb)	$t_{swarb}(p) = 21\ (1/2) \log_4 p + 14(1/12)$ $h_{swarb}(p) = 9$	9.6	9.9
Crossbar traversal (xbar)	$t_{xbar}(p, w) = 9 \log_8 (w \lfloor p/2 \rfloor) + 6 \lceil \log_2 p \rceil + 6$ $h_{xbar}(p, w) = 0$	8.4	10.5
Virtual-channel router			
Virtual-channel allocator (vcalloc)	$t_{vcalloc}(p, v) = 33 \log_4 pv + 20(5/6)$ $h_{vcalloc}(p, v) = 9$	16.9	15.3
Switch allocator (swalloc)	$t_{swalloc}(p,v) = 11(1/2) \log_4 p + 23 \log_4 v + 20(5/6)$ $h_{swalloc}(p,v) = 9$	10.9	12.0
Crossbar traversal (xbar)	$t_{xbar}(p, w) = 9 \log_8 (w \lfloor p/2 \rfloor) + 6 \lceil \log_2 p \rceil + 6$ $h_{xbar}(p, w) = 0$	8.4	10.5

the sum of the path's effort delay (T_{eff}) and parasitic delay (T_{par}). The effort delay of each stage is the product of logical effort and electrical effort. Logical effort is the ratio of a logical function's delay to the delay of an inverter with identical input capacitance. Electrical effort is the fan-out, or the ratio of output capacitance to input capacitance. Parasitic delay is the intrinsic delay of a gate due to its own internal capacitance and is expressed relative to an inverter's parasitic delay.

$$T = T_{eff} + T_{par} = \sum g_i h_i + \sum p_i$$

where g_i is the logical effort per stage, h_i is the electrical effort per stage, and p_i is the parasitic delay per stage.

Table 1 lists the technology-independent parameterized delay equations for each atomic module. The gate-level circuit designs of each module and the analysis and derivation of these parameterized delay equations are presented in detail in another publication.[12] In brief, we used separable allocators and the matrix arbiter. We validated projections of the model against a projection estimated by a Synopsys timing analyzer in a 0.18-micron technology and found them to be close—within $2\tau_4$.

Pipeline latency results

Figure 5 (next page) details the effect of differing numbers of physical and virtual channels on the per-node latency of a virtual-channel router, with a typical clock cycle of $20\tau_4$ (τ_4 is the delay of an inverter driving four other inverters[11]). For a two-dimensional virtual-channel router with five physical channels, four pipeline stages are sufficient for up to eight virtual channels per physical channel. Thus, overall per-node latency doesn't increase as the number of virtual channels increases from two to eight. With a three-dimensional network, a virtual-channel router with seven physical channels incurs the same per-node latency of four cycles (from its four pipeline stages) for up to eight virtual channels per physical channel.

These projections indicate that a virtual-channel router typically requires just one more pipeline stage than a wormhole router (a speculative virtual-channel router can have the same per-hop latency as a wormhole router). Also, with most practical numbers of virtual channels used in virtual-channel routers to date, the delay remains unchanged.

Simulation results

Guided by the pipelined designs prescribed by the model, we crafted detailed Verilog code for wormhole and virtual-channel routers and performed simulations to determine their latency-throughput characteristics. Again, we assumed a typical clock cycle of $20\tau_4$. The simulator generated uniformly distributed traffic across an eight-by-eight mesh network to random destinations. Each simulation ran for a warm-up phase of 10,000 cycles. Then, we

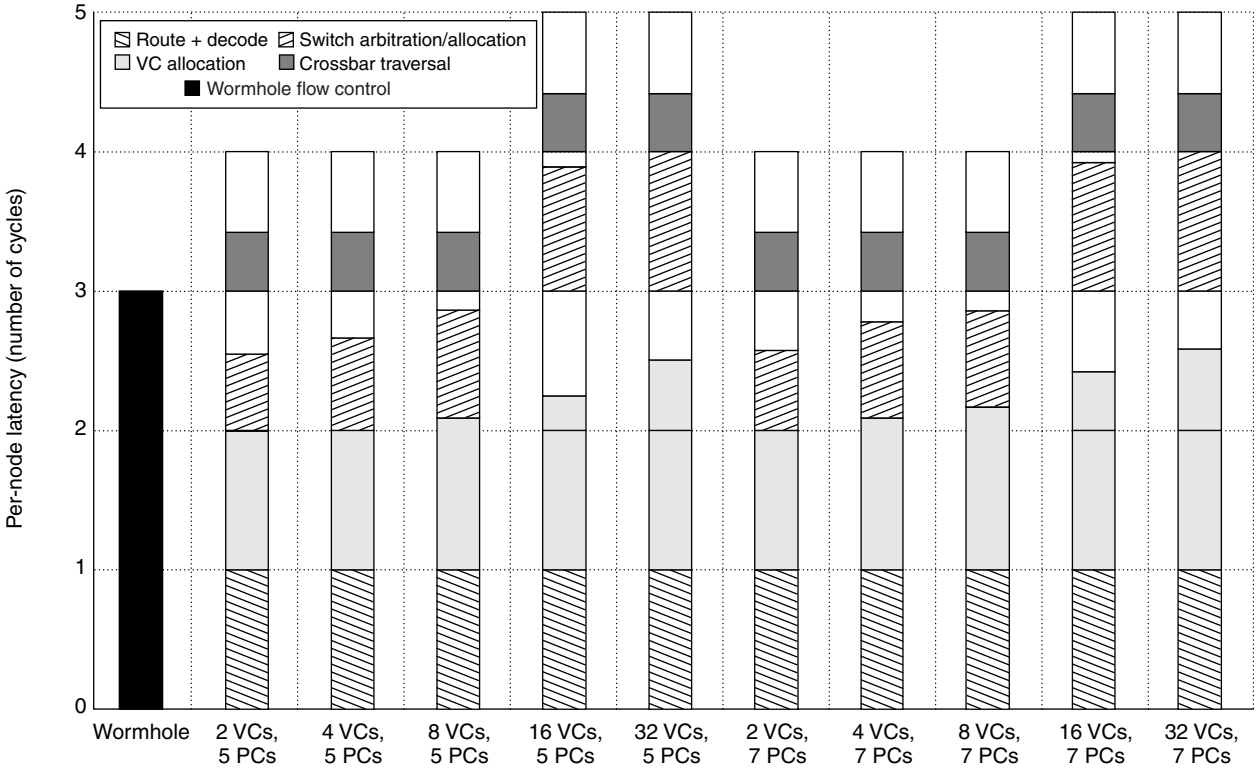

Figure 5. Effect of p, the number of physical channels (PCs), and v, the number of virtual channels (PCs), on per-node latency of virtual-channel routers. We assumed a typical clock cycle of $20\tau_4$. Each bar represents the router pipeline, with the shaded regions corresponding to the fraction of clock cycle time used by each atomic module and the idle time left unshaded.

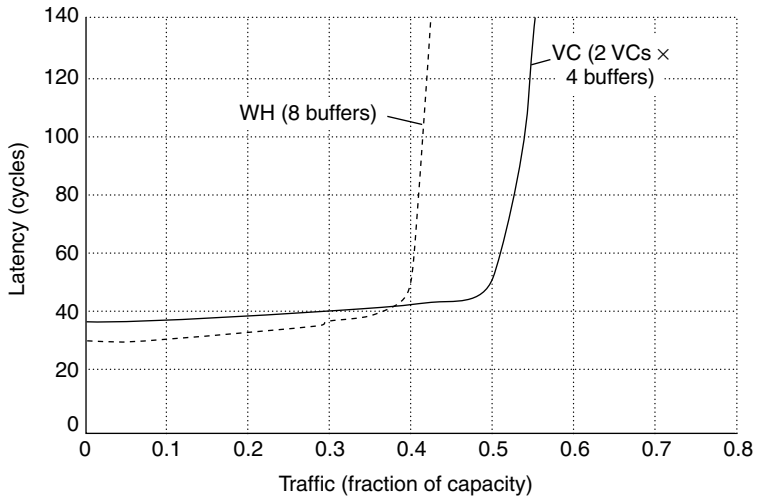

Figure 6. Latency-throughput curves of wormhole and virtual-channel routers with eight buffers per input port. The routers follow the proposed pipeline router model.

injected 100,000 packets and ran the simulation until these packets were all received. Throughout the simulation, a constant-rate source injects five-flit packets at a percentage of the network's capacity, and the average latency of a packet is calculated. Latency of a packet begins the instant the packet's first flit is created and ends when its last flit is ejected at the destination node, including source queuing time and assuming immediate ejection. Each router uses credit-based flow control to regulate buffer use, and propagation delay across the channel is assumed to take a single cycle. Because the purpose of our simulations was to explore the performance of flow control strategies, we chose simple dimension-ordered routing.

Performance comparisons

Figure 6 shows the latency-throughput curves of wormhole and virtual-channel routers with eight buffers per input port. The wormhole router uses a three-stage pipeline, whereas the virtual-channel router is pipelined over four stages at each hop. Therefore, the

base latency of the wormhole router (29 cycles) is lower than that of the virtual-channel router (36 cycles), However, virtual-channel flow control with two virtual channels extends the throughput achieved by wormhole flow control by 25% from 40% capacity to 50% capacity.

With 16 buffers per input port, we observed similar effects, shown in Figure 7. Again, a virtual-channel router has a higher base latency (35 cycles) than that of a wormhole router, as a result of the additional pipeline stage per hop. However, the throughput advancement is large; a virtual-channel router with two virtual channels enjoys a throughput of 65% capacity, a 30% improvement over the wormhole router's 50% capacity. With four virtual channels, throughput increases to 70% capacity, a 40% improvement over wormhole flow control.

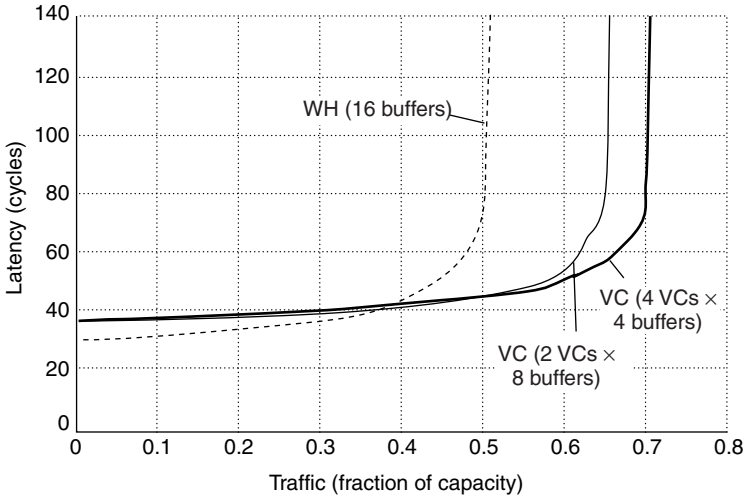

Figure 7. Latency-throughout curves of wormhole and virtual-channel routers with 16 buffers per input port. The routers are pipelined as prescribed by the proposed pipelined router model.

Single-cycle router latency

Most published research comparing the performance of different router designs assumes single-cycle router latency, without taking into account the effect of implementation complexity and cost. To quantify this effect, we ran simulations with a cycle-accurate C simulator that assumes single-cycle router latency for both wormhole and virtual-channel flow control. All other experimental parameters were identical to those of the Verilog simulator.

As Figure 8 shows, assuming single-cycle router latency results in both wormhole and virtual-channel routers' incurring the same low base latency of 16 cycles. In contrast, simulations that follow our proposed multiple-cycle pipeline design demonstrate the higher base latency incurred by virtual-channel flow control because of its longer pipeline.

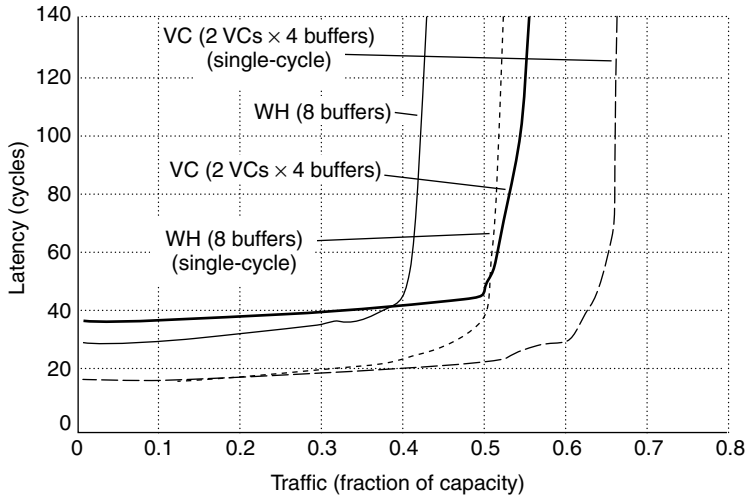

Figure 8. Performance of wormhole and virtual-channel routers, as modeled by the proposed pipelined delay model, and as modeled with single-cycle router delay (eight buffers per input port).

It is also apparent that assuming single-cycle router latency results in inflated throughput figures. This is because throughput in wormhole and virtual-channel flow control is strongly influenced by buffer utilization. Buffer utilization depends on how quickly the router can send and receive credits, prompting the reuse of buffers. In the simulations assuming single-cycle routing latency, a credit can be sent and received in two cycles. In our pipelined model, a wormhole router needs four cycles for credit turnaround, and a virtual-channel router needs five cycles. Thus, throughput is lower in the pipelined model than in one that ignores implementation delay.

Motivated by the proposed delay model, we uncovered a more efficient microarchitecture for virtual-channel routers, using speculation to shorten the critical path.[9] This reduces a virtual-channel router's latency to that of a wormhole router, and further increases its throughput.

The proposed delay model can be extend-

ed in many interesting directions. Whereas the model currently supports only wormhole and virtual-channel flow control, we are in the process of applying it to flit-reservation flow control.[13] It can also be expanded to cover other topologies and routing policies, such as adaptive routing. While router latency holds the key to network performance, a router's area or power dissipation may be critical for certain applications of interconnection networks. Hence, incorporating area and power projections into our delay model will be exciting future work. MICRO

References

1. J. Duato, S. Yalamanchilli, and L. Ni, *Interconnection Networks—An Engineering Approach*, IEEE CS Press, Los Alamitos, Calif., 1997.
2. M. Galles, "Spider: A High-Speed Network Interconnect," *IEEE Micro*, vol. 17, no. 1, Feb. 1997, pp. 34-39.
3. Cray Inc., http://www.cray.com/products/systems/crayt3e/paper1.html.
4. N.J. Boden et al., "Myrinet—A Gigabit-per-Second Local-Area Network," *IEEE Micro,* v. 15, no. 1, Feb. 1995, pp. 29-36.
5. Avici Systems Inc., http://www.avici.com.
6. A.A. Chien, "A Cost and Speed Model for k-ary n-Cube Wormhole Routers," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 2, Feb. 1998, pp. 150-162.
7. W.J. Dally and C. Seitz, "The Torus Routing Chip," *Distributed Computing*, vol. 1, no. 3, 1986, pp. 187-196.
8. J. Duato and P. Lopez, "Performance Evaluation of Adaptive Routing Algorithms for k-ary n-Cubes," *Proc. Parallel Computer Routing and Communication Workshop, Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1994, pp. 45-59.
9. L.-S. Peh and W.J. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers," to be published in *Proc. Seventh Int'l Symp. High-Performance Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., 2001.
10. R.F. Sproull and I.E. Sutherland, "Logical Effort: Designing for Speed on the Back of an Envelope," *IEEE Advanced Research in VLSI,* C. Sequin, ed., MIT Press, Cambridge, Mass., 1991, pp. 1-16.
11. W.J. Dally and J.W. Poulton, *Digital Systems Engineering*, Cambridge University Press, Cambridge, U.K., 1998.
12. L.-S. Peh, *Flit-Reservation Flow Control*, PhD thesis, Dept. of Computer Science, Stanford Univ., Stanford, Calif., to be published in 2001.
13. L.-S. Peh and W.J. Dally, "Flit-Reservation Flow Control," *Proc. Sixth Int'l Symp. High-Performance Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 73-84.

Li-Shiuan Peh is a PhD candidate in computer science at Stanford University. Her interests are high-speed interconnection networks and communication systems. She received a BS in computer and information systems from the National University of Singapore. She is a student member of the IEEE.

William J. Dally is a professor of electrical engineering and computer science at Stanford University, where his group developed the Imagine processor, which introduced the concepts of stream processing, partitioned register organizations, and low-power, high-speed signaling technology. Earlier, he was a professor at the Massachusetts Institute of Technology, where he and his group built the J-Machine and the M-Machine, experimental parallel computer systems. As a research assistant and research fellow at Caltech, he designed the MOSSIM Simulation Engine and the Torus Routing Chip, which pioneered wormhole routing and virtual-channel flow control. Dally received the BS in electrical engineering from Virginia Polytechnic Institute, the MS in electrical engineering from Stanford University, and the PhD in computer science from Caltech. He is a member of the IEEE, the Computer Society, the Solid-State Circuits Society, and the ACM.

Send questions and comments about this article to Li-Shiuan Peh, Stanford University, Gates Building 2A, Room 212, Stanford, CA 94305; lspeh@cs.stanford.edu.