In Proceedings of the 7th International Symposium on High-Performance Computer Architecture, Jan. 22-24, 2001, Monterrey, Mexico, pp. 255-266.
(Best Student Paper Award)

A Delay Model and Speculative Architecture for Pipelined Routers

Li-Shiuan Peh
lspeh@cs.stanford.edu

William J. Dally
billd@csl.stanford.edu

Computer Systems Laboratory
Stanford University
Stanford, CA94305

Abstract

This paper introduces a router delay model that accurately models key aspects of modern routers. The model accounts for the pipelined nature of contemporary routers, the specific flow control method employed, the delay of the flow-control credit path, and the sharing of crossbar ports across virtual channels. Motivated by this model, we introduce a microarchitecture for a speculative virtual-channel router that significantly reduces its router latency to that of a wormhole router. Simulations using our pipelined model give results that differ considerably from the commonly-assumed 'unit-latency' model which is unreasonably optimistic. Using realistic pipeline models, we compare wormhole [6] and virtual-channel flow control [4]. Our results show that a speculative virtual-channel router has the same per-hop router latency as a wormhole router, while improving throughput by up to 40%.

1. Introduction

Interconnection networks are used to connect processors to memories in multicomputers and multiprocessors and to connect line cards in network switches and Internet routers. The performance of interconnection networks and hence of the systems in which they are employed depends critically on the performance of the routers from which these networks are constructed. Accurate performance models are needed to enable the architecture of these networks to be optimized.

This paper introduces a router delay model that accurately models key aspects of modern routers. This model accounts for the pipelined nature of contemporary routers, the specific flow-control method employed, the delay of the flow-control credit path, and the sharing of crossbar ports across virtual channels. Our model uses technology-independent parametric equations for delay that are derived from detailed gate-level designs and analyses.

Motivated by our model, we introduce a microarchitecture for a *speculative* virtual-channel router. In a conventional virtual-channel router, an arriving packet must first arbitrate for an output virtual channel (VC) before arbitrating for switch bandwidth. This serialization of resource arbitration significantly increases latency compared to a wormhole router. A speculative virtual-channel router arbitrates for an output VC and switch bandwidth in parallel, speculating that it will be allocated a VC. If the speculation turns out to be incorrect, the crossbar passage is simply wasted. As the switch allocator prioritizes non-speculative requests over speculative ones, there is no adverse impact on throughput.

We develop canonical pipelines for wormhole routers, virtual-channel routers, and speculative virtual-channel routers. For each of these pipelines, our gate-level models are used to provide accurate estimates of delay and to determine pipeline depth as a function of cycle time. Using this model we compare wormhole and virtual-channel flow control. Our results show that a speculative virtual-channel router can achieve the same zero-load latency as a wormhole router but with 40% higher throughput. Compared to the commonly assumed 'unit-latency' model, our model shows substantial performance differences of 56% in zero-load latency, and 30% in throughput. This establishes the need to account for pipeline delays and credit latency in performance models.

The paper next discusses current router models and addresses their inaccuracies in Section 2. Section 3 delves into the details of the proposed router delay model and Section 4 applies the model to explore the effect of varying numbers of physical and virtual channels on the latency of a pipelined router. Simulation results comparing wormhole and virtual-channel routers using pipelines proposed by the model are presented in Section 5 and Section 6 concludes the paper.

2. Related Work

Chien [2, 3] first noted the need for router delay models which consider implementation complexity, and proposed a router model for wormhole and virtual-channel routers.

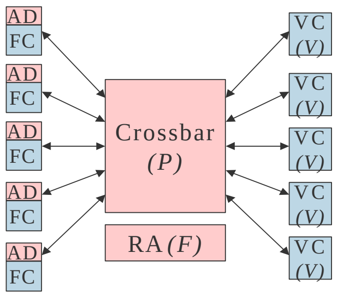


Figure 1. Canonical router architecture proposed in Chien's model. The parameters of the delay model are *P*, the number of ports on the crossbar; *F*, the number of output route choices; and *V*, the number of virtual channels per physical channel.

Chien's model uses the router architecture of Figure 1, which was employed in the Torus Routing Chip [6], for all routers regardless of the flow control method employed. It defines per-hop router latency as the total delay of the functions on the critical path through address decoding, routing, crossbar arbitration, crossbar traversal, and virtual channel allocation. Through detailed gate-level design and analysis, the delay of these functions are expressed in parameterized equations which are then 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.

Chien's model, however, has several significant shortcomings. First, it does not account for pipelining, assuming instead that the entire critical path fits within a single clock cycle. Second, this model assumes that the crossbar must provide a separate port for each virtual channel. This causes the crossbar and its arbitration and traversal latency to grow very rapidly with the number of virtual channels. Most real routers, on the other hand, share one or a few crossbar ports across the virtual channels associated with a single physical channel.

Duato [7] extended Chien's model to consider a fixed three-stage pipeline. The pipeline consists of a routing stage, that contains the address decode, routing, and arbitration functions of Chien's model; a switching stage that includes the crossbar traversal, and a channel stage that includes the virtual channel allocation and the inter-node delay.

Miller and Najjar extended Chien's model for virtual cut-through routers, modifying the parameterized delay equation for flow control to include the parameter B, the number of buffers in that input queue [8].

These models all assume that clock cycle time depends solely on router latency. In practice, however, routers are heavily pipelined, making cycle time largely independent of router latency. Typically, router designers have to work within the limits of a clock cycle that is determined by factors beyond the router, such as the fundamental limits of chip-to-chip signalling [1], 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 variable.

These previous models also attempt to fit all routers into a single canonical architecture. This results in a mismatch between the architecture and the flow control method. For instance, passage through the crossbar is arbitrated on a per-packet basis and held throughout the duration of a packet, prompting the need for a huge crossbar in a virtual-channel router, with the number of ports equal to the total number of virtual channels. This contributes unnecessary delay in the crossbar arbitration and traversal functions. Buffering flits at virtual channel controllers whose arbitration delay increases with the number of virtual channels also adds needless cost to a virtual-channel router.

3. Proposed model

The proposed router delay model comprises a *general router model* which outlines a design methodology for the pipelining of a router given a clock cycle time, using the delay estimates derived by a *specific router model* to prescribe a pipeline design. We will walk through its application to wormhole and virtual-channel routers, starting with the definition of canonical router architectures which are suited for each flow control method, proceeding on to uncover the atomic modules of each flow control and the dependencies between them, before deriving parametric delay equations for each atomic module using the specific router model. These equations provide delay estimates which are used to arrive at a realistic pipeline design.

3.1 General router model

Canonical router architectures. Figure 2 illustrates the canonical wormhole router architecture[1]. As shown, the parameters affecting the delay of the various modules of a wormhole router are *p*, the number of physical channels, and *w*, the channel width or phit size.

Consider a two-flit packet, 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*[2], *switch*

1. This is similar to the canonical router architecture proposed in Chien's model.
2. Throughout the paper, our emphasis is on a comparison of flow control techniques. Hence, we will view routing as a black box and assume decoding and routing takes a typical clock cycle of 20 τ_4, where τ_4 is the delay of an inverter driving 4 other inverters [5].

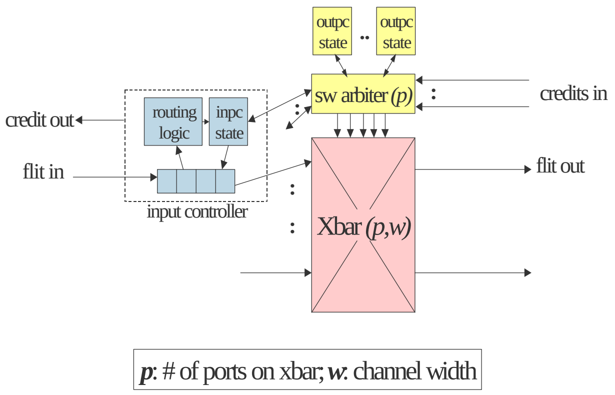


Figure 2. Canonical wormhole router architecture of the proposed model.

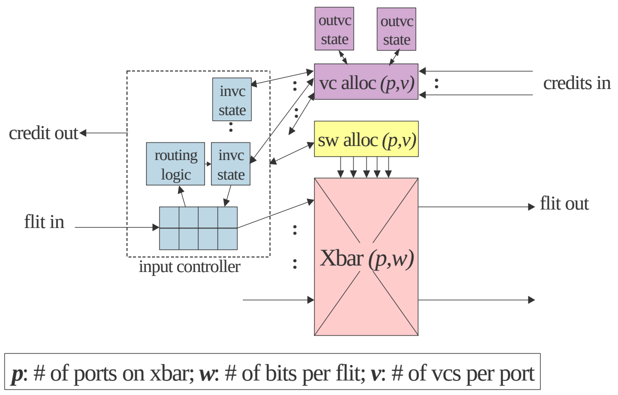


Figure 3. Canonical virtual-channel router architecture of the proposed model.

arbitration, and *switch traversal*. When the head flit arrives, the input controller decodes its type field and finding it a head flit forwards the destination field to the routing logic, buffers the entire flit in the input queue, and sets the channel state (*inpc_state*) to *routing*.

The routing logic returns the output port[3] for this packet, in this case the eastern output port. At this point, the channel state is set to *switch arbitration,* and a request for the eastern output port is sent to the *global switch arbiter*, which receives arbitration requests from all input controllers and assigns available output ports to the requestors, resolving contention. In this case, the global switch arbiter grants the eastern output port to the request from the injection input channel, and flags this output port as unavailable to other requests.

Upon receipt of this grant from the global switch arbiter, the state of the injection input channel is set to *switch traversal*, and the head flit of the packet is read from the input queue, and input to the *crossbar*. The crossbar configuration is set by the switch arbiter, which in this example, has connected the injection input port to the eastern output port. The head flit hence traverses the crossbar and proceeds over the output link for the next hop.

When the next flit arrives at the input controller, its type field is again decoded and found to be a tail flit. Since it is not a head flit, it need not go through routing or switch arbitration. Instead, it is simply buffered in the input queue and sent to the output port reserved by the head flit. When the tail flit departs the input queue, it releases the resources held by the packet by setting the input state to *idle* and signaling the global switch arbiter to free up the reserved output port.

3. We will assume a deterministic routing algorithm here. If it is an adaptive routing algorithm, more than one output port may be supplied.

Figure 3 shows the canonical router architecture for virtual-channel flow control. The additional parameter here is *v*, the number of virtual channels per physical channel. This architecture differs from the canonical router architecture 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 on a flit-by-flit basis.

Consider the same two-flit packet flowing through a virtual-channel router. The packet proceeds as above but with the addition of a *virtual channel allocation* state. In the virtual-channel router, there is a separate input queue and a separate copy of the channel state (*invc_state*), for each virtual channel. When the head flit of this packet arrives at the *input controller* of the injection channel, its virtual-channel id (vcid) field is decoded and the entire flit is buffered in the appropriate flit queue. For instance, the packet in our example is injected into virtual channel 0 (VC 0) of the injection channel, and buffered accordingly into queue 0. At this point, VC 0 enters the *routing* state, and the destination field of the flit is sent to the *routing logic*, which returns the output virtual channels (not physical channels) the packet may use.

Upon receipt of the output virtual channels, VC 0 state is set to *virtual channel allocation*. VC 0 then requests its desired output virtual channels from the *global virtual channel allocator*, which collects all the requests from each VC of the input controllers and returns available output virtual channels to successful requestors. It also updates the status of these output virtual channels as unavailable in *outvc_state*.

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*. Instead of reserving output ports for the entire duration of a packet, the switch allocator of a virtual-channel router allocates crossbar passage to flits of differ-

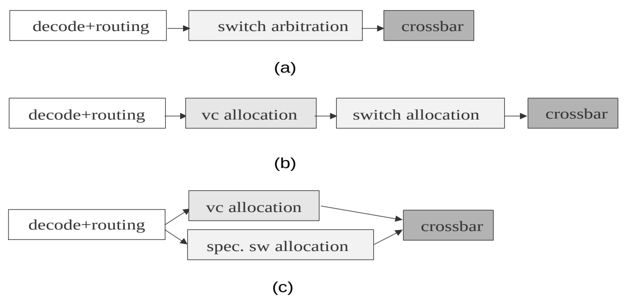


Figure 4. Atomic modules and dependences of (a) a wormhole router; (b) a virtual-channel router; (c) a speculative virtual-channel router.

ent packets on a cycle-by-cycle basis. Once this head flit secures passage through to the eastern output port, it leaves for the *crossbar* and to the next hop, with its vcid field updated to VC 1.

When the subsequent tail flit arrives, it is enqueued into the appropriate queue based on 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 it is granted crossbar passage, it informs the virtual-channel allocator to release the reserved output VC, and leaves for the next hop, with its vcid field also updated to VC 1. Note that in this canonical architecture, the function of multiplexing virtual channels onto a physical channel rests upon the switch allocator, instead of the virtual channel controllers of Chien's architecture.

Atomic modules and dependencies. The canonical router architectures shown in Figures 2 and 3 include a number of modules that are not easily pipelined because they contain state that is dependent on the module output. These modules are termed *atomic modules* in our model, and 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, it can result in grants not being reflected correctly before the next allocation[4]. Also, with a separable allocator, there are a large number of wires connecting the input and output ports, which will require excessive latching should the allocator be partitioned across multiple pipeline stages. Figure 4 shows the various atomic modules of wormhole and virtual-channel routers in our model.

The inputs of these atomic modules may depend on the outputs of another, in which case, a dependency exists. These dependencies determine the critical path of a router. Figure 4(a) and (b) shows the basic dependencies of a wormhole and virtual-channel router respectively.

These dependencies can sometimes be averted with speculation. 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, and thus, proceed to request the desired output port[5] before it has secured an output virtual channel. Should the speculation be incorrect, the crossbar passage reserved will just 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 shown in Figure 4(c). To avoid any impact on throughput, the speculative switch allocator needs to prioritize non-speculative requests over speculative ones. Note that as the switch is arbitrated on a cycle-by-cycle basis, and is not held indefinitely by a packet, there is no possibility of a deadlock as a result of switch allocation being performed speculatively with virtual-channel allocation.

For the two-flit packet example in a speculative virtual-channel router, while the head flit is submitting its requests for output virtual channels to the global virtual-channel allocator, it will at the same time send its request for the eastern output port to the global switch allocator. If no other non-speculative packets are requesting for the eastern output port and this packet wins the arbitration, the switch allocator will grant the eastern output port to the head flit, which will leave for the crossbar once it is also granted an output virtual channel. The subsequent tail flit will still send requests to the switch allocator, but its request will be a non-speculative one since it has already inherited the output VC reserved by its head flit.

Pipeline design. The delay of each atomic module is modelled by the parametric equations derived by the specific router model (Section 3.2) which generates two delay estimates: *latency* (t_i) and *overhead* (h_i). Latency spans from when inputs are presented to the module, to when the outputs needed by the next module are stable. Overhead refers to the delay expended by additional circuitry required before the next set of inputs can be presented to

4. There are many different ways to pipeline a virtual-channel allocator should it require more than a single clock cycle. One possibility is to pessimistically update all requested virtual channels as taken so that the first level of arbitration can continue at the next cycle. Once the actual granted virtual channels are known, the status is then brought up-to-date. As with most ways of pipelining an atomic module, performance may suffer.

5. For a deterministic router, there is a single desired output port. Thus, the objective of the speculative switch allocator is straight-forward: bid for this output port. In an adaptive router, the speculative switch allocator can either bid for all desired output ports returned by the routing function, and only permit crossbar passage if the output port granted matches that granted by the virtual-channel allocator; or the routing function may be limited to returning only a single output port ($R \rightarrow p$), and packets reiterates through the routing process upon failure in obtaining a free virtual channel.

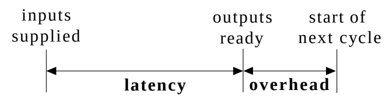


Figure 5. Latency and overhead estimates derived by the specific router model.

the module. Figure 5 shows these delay components. In a switch arbiter, for instance, latency spans from when requests for switch ports are presented to when grant signals are stable, while overhead refers to the delay in the arbiter for updating the priorities between requestors 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 pipelining of the router as follows :-

Given a, the first atomic module and

b, the last atomic module in the pipeline stage,

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

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

...(EQ 1)

3.2 Specific router model

Logical effort. We calculate the delays of each atomic module using the method of logical effort [10, 11]. Using this method, the circuit delay, T (in τ), along a path is calculated as the sum of the *effort delay* (T_{eff}) and the *parasitic delay* (T_{par}) of that path where the effort delay of each stage is the product of logical effort and electrical effort (EQ 2). Logical effort is the ratio of a logical function's delay to the delay of an inverter with identical input capacitance[6]. Electrical effort is the fanout, the ratio of output capacitance to input capacitance. Parasitic delay refers to the intrinsic delay of a gate due to its own internal capacitance, and is expressed relative to the parasitic delay of an inverter.

$$\begin{aligned} T &= T_{eff} + T_{par} \\ &= \sum g_i h_i + \sum p_i \end{aligned}$$

$$\begin{aligned} \text{where } g_i &= \text{logical effort per stage} \\ h_i &= \text{electrical effort per stage} \\ p_i &= \text{parasitic delay} \end{aligned}$$

...(EQ 2)

6. The delay of an inverter with identical input capacitance is τ [5].

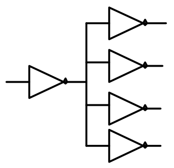


Figure 6. An inverter driving four other inverters. Its delay, τ_4, is derived using the method of logical effort to be 5τ.

As an example, we apply the method of logical effort to derive the delay of an inverter driving four other inverters, shown in Figure 6. Its electrical effort, h_i, is 4 since output capacitance is 4 times input capacitance; and both logical effort g_i and parasitic delay p_i are 1 given they are defined relative to an inverter (EQ 3). Thus, the delay of an inverter driving 4 other inverters, $\tau_4 = 5\tau$.

$$\begin{aligned} T_{eff} &= \sum g_i h_i = 1 \times 4 \\ T_{par} &= 1 \\ T = T_{eff} + T_{par} &= 4 + 1 = 5 \end{aligned}$$

...(EQ 3)

Design of switch and virtual-channel allocators. The switch and virtual channel allocators match resource requests with free resources. Our model assumes a separable design of each of these allocators in which requests for a given resource from a single input port are arbitrated in the first stage of the allocator. The winning request from each input port then arbitrates for the resource in the second stage. Separable allocators admit a simple implementation while sacrificing a small amount of allocation efficiency compared to more complex approaches.

As the switch in a wormhole router is held throughout the duration of a packet, status needs to be kept for each output port, as shown in Figure 7(a)[7]. In a virtual-channel router, on the other hand, the switch is allocated on a cycle-by-cycle basis and no state needs to be stored. Figure 7(b) shows the switch allocator of a non-speculative virtual-channel router, with a *v:1* arbiter in the first stage determining which virtual channels of an input port gets to bid for its output port, and a p_i*:1* arbiter for each output port in the second allocation stage. In a speculative virtual-channel router, the switch allocator needs to ensure non-speculative requests have higher priority than speculative ones. A way to achieve this is shown in Figure 7(c), where two switch allocators handle non-speculative and speculative requests in parallel, and successful non-speculative requests are selected over speculative ones.

7. We assume a deterministic routing algorithm for the wormhole router, since adaptive wormhole routers will need to be deadlock-free without using virtual channels.

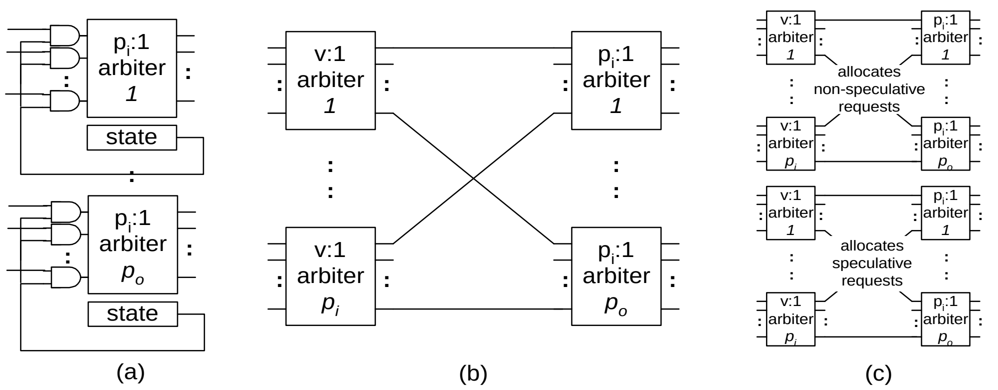


Figure 7. (a) A switch arbiter for a wormhole router, with a p_i:1 arbiter for each output port. Since a wormhole router holds the switch for the duration of a packet, state needs to be stored reflecting the status of each output port. Updating of state is not on the critical path, and is thus not detailed for brevity. (b) A separable switch allocator for a non-speculative virtual-channel router, with the first stage of v:1 arbiters for each input port, and a second stage of p_i:1 arbiters for each output port. For simplicity, the diagram shows the outputs of v:1 arbiters connected to the inputs of p_i:1 arbiters, while in reality, the outputs of the v:1 arbiters select the output port request of the winning VC and forward it on to the second stage of p_i:1 arbiters. (c) Separable switch allocators for a speculative virtual-channel router, one handling non-speculative requests, and another handling speculative requests. Non-speculative requests have higher priority over speculative ones. (p_i, number of input switch ports, and p_o, number of output switch ports = p, the number of ports in the router; v is the number of virtual channels per physical channel)

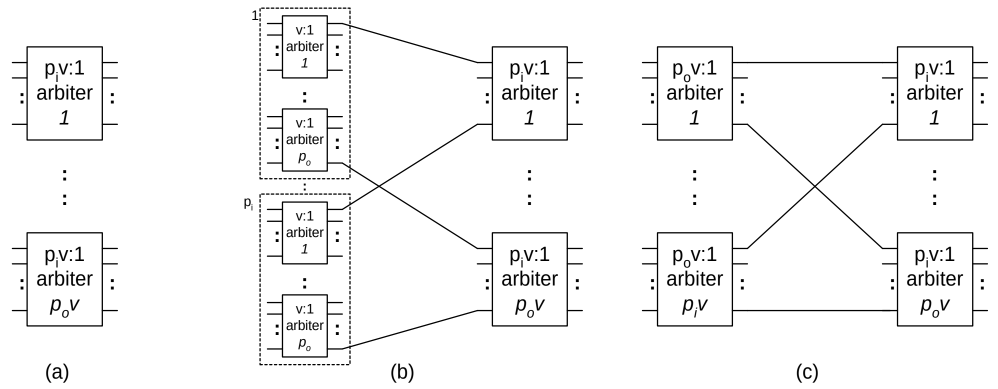


Figure 8. Complexity of a virtual-channel allocator, given routing functions with different ranges. (a) Given a routing function which returns a single virtual channel ($R \to v$), the virtual-channel allocator only needs a p_iv:1 arbiter for each output virtual channel. (b) Given a routing function which returns virtual channels of a single physical channel ($R \to p$), the virtual-channel allocator needs the first stage of v:1 arbiters for each input virtual channel, followed by a second stage of p_iv:1 arbiters for each output virtual channel. (c) Given the most general routing function which returns candidate virtual channels of any physical channels ($R \to pv$), the first stage of a virtual-channel allocator needs a p_ov:1 arbiter to handle the maximum p_ov output virtual channels desired by each input virtual channel, followed by a p_iv:1 arbiter for each output virtual channel. (p_i, number of input switch ports, and p_o, number of output switch ports = p, the number of physical ports in the router; and v is the number of virtual channels per physical channel)

The complexity and latency of a virtual-channel allocator of a virtual-channel router depends on the range of the routing function. If the routing function returns a single virtual channel *($R \to v$)*, the virtual-channel allocator needs only arbitrate among input virtual channels which are competing for the same output virtual channel, as in Figure 8(a). If the routing function is more general and returns any candidate virtual channels of a single physical channel *($R \to p$)*, each of the arbiters now needs to arbitrate among v possible requests in the first stage of the separable allocation, before forwarding to the arbiters in the second stage, as in Figure 8(b). Using these restricted routing functions require iterating through the routing function upon each unsuccessful bid[8]. If we use the most general

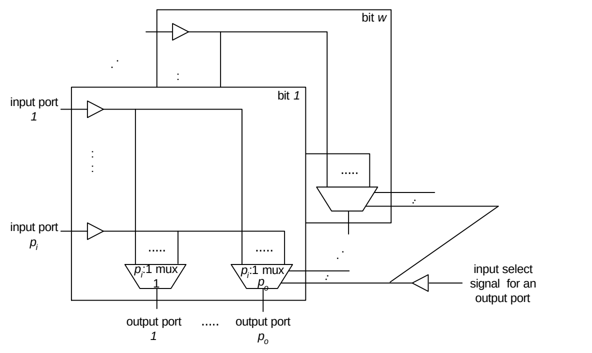


Figure 9. A diagram of the gate-level design of a crossbar with p_i input ports and p_o output ports, each *w* bits wide.

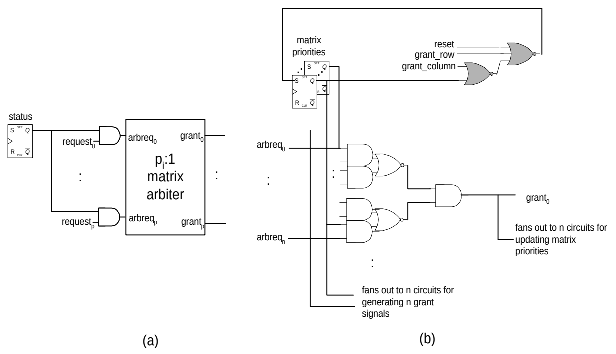


Figure 10. A sketch of the gate-level design of a switch arbiter in a wormhole router. (a) shows the switch arbiter for a single output port, which consists of a p_i:1 matrix arbiter and a flip-flop storing the status of the output port. (b) shows the key components of the matrix arbiter, with the circuitry contributing to t_{SB} unshaded, and the circuitry contributing to h_{SB} shaded in grey.

routing function which returns all possible candidate virtual channels of all physical channels $(R \rightarrow pv)$, this heaps on more responsibility onto the virtual-channel allocator, which now needs to go through two stages of *pv:1* arbiters on its critical path, as shown in Figure 8(c).

Design of crossbar. Figure 9 shows the design of a *p*-port crossbar, with each port *w* bits wide. Select signals direct the connections between input and output ports, and originate from the switch allocator. Critical-path delay comprises the fan-out of select signals to the multiplexers in the *w* bit-slices of the crossbar and the delay through the multiplexers. It should be noted that the proposed model does not take into account wire delay, which is a significant delay component in the crossbar. To alleviate the impact this has on simulation results, the crossbar pipeline stage is kept as a single pipeline stage, assuming it requires all 20 τ_4. This is a reasonable assumption for the small crossbar sizes considered in this paper, as traversal, including wire delays, will fit comfortably within 20 τ_4[9].

Derivation of parametric delay equations for a switch arbiter of a wormhole router. This arbiter[10] consists of p_o p_i*:1* arbiters, one for each output port. State is maintained for each output port to indicate if that port is already held by a packet and hence unavailable for a new request. Each p_i*:1* arbiter is implemented using an upper triangular p_i *x* p_i matrix of flip-flops that records the binary priority between each pair of inputs. A requestor wins if it has a higher priority than all other requestors of that resource. Once a requestor succeeds in an arbitration, its priority is set to be the lowest among all requestors. The gate-level design of the switch arbiter is sketched in Figure 10. Note that the updating of the status flip-flops can occur during the arbitration and does not contribute to h_{SB}, since an output port will be granted as long as it has at least one request.

The derivations of the effort delay ($T_{eff_{arb}}$) and parasitic delay ($T_{par_{arb}}$) of a n:1 matrix arbiter are shown in (EQ 4) and the derivations of the latency (t_{SB}) and overhead (h_{SB}) equations of the switch arbiter atomic module are shown in (EQ 5) and (EQ 6) respectively :-

$$
\begin{aligned}
T_{eff\,arb}(n) = {} & \frac{6}{3}_{(aoi)} + \\
& \frac{\log_4(\frac{n}{2})}{2}\left(\frac{6}{3}+\frac{9}{3}\right)_{(\frac{n}{2}-input\ and:\log tree\ of\ nands\ and\ nors)} + \\
& 4\log_4 n_{(fanout\ to\ n\ circuits\ for\ updating\ matrix\ priorities)} \\
= {} & 6\frac{1}{2}\log_4 n + \frac{3}{4}
\end{aligned}
$$

$$
\begin{aligned}
T_{par\,arb}(n) = {} & 8_{(aoi)} + \\
& \frac{\log_4(\frac{n}{2})}{2}(4+4)_{(\frac{n}{2}-input\ and:\log tree\ of\ nands\ and\ nors)} + \\
& \log_4 n_{(fanout\ to\ n\ circuits\ for\ updating\ matrix\ priorities)} \\
= {} & 5\log_4 n + 6
\end{aligned}
$$

...(EQ 4)

8. Note that for a deterministic router, the routing function which returns candidate virtual channels of a single physical channel ($R \rightarrow p$) is the most general possible.

9. This assumption may not hold for the crossbars in Chien's model which require *pv* ports instead of *p* ports. In these cases, crossbar traversal may have to be pipelined into multiple stages.

10. We assume an arbiter for wormhole routers, because we are assuming the routing function returns a single output port. This is reasonable even for adaptive routing functions, which can iterate and return another output port should the arbitration fails. Besides, it makes no difference to performance as we are using separable allocators.

$$T_{eff_{t_{SB}}}(p) = 2_{(status\ latch)} + 4\log_4 p_{(fanout\ to\ p\ requests)} + \frac{4}{3}_{(2-input\ nand)} + 4\log_4 p_{(fanout\ to\ p\ grant\ circuits\ in\ the\ arbiter)} + T_{eff_{arb}}(p)_{(p:1\ matrix\ arbiter)} = 14\frac{1}{2}\log_4 p + 4\frac{1}{12}$$

$$T_{par_{t_{SB}}}(p) = 2_{(status\ latch)} + \log_4 p_{(fanout\ to\ p\ requests)} + 2_{(2-input\ nand)} + \log_4 p_{(fanout\ to\ p\ grant\ circuits\ in\ the\ arbiter)} + T_{par_{arb}}(p)_{(p:1\ matrix\ arbiter)} = 7\log_4 p + 10$$

$$\therefore t_{SB}(p) = T_{eff_{t_{SB}}}(p) + T_{par_{t_{SB}}}(p) = 21\frac{1}{2}\log_4 p + 14\frac{1}{12}$$

...(EQ 5)

$$T_{eff_{h_{SB}}}(p) = \frac{5}{3}_{(2-input\ nor)} + \frac{7}{3}_{(3-input\ nor)} = 4$$

$$T_{par_{h_{SB}}}(p) = 2_{(2-input\ nor)} + 3_{(3-input\ nor)} = 5$$

$$\therefore h_{SB}(p) = T_{eff_{h_{SB}}}(p) + T_{par_{h_{SB}}}(p) = 9$$

...(EQ 6)

Through detailed gate-level design and analysis of the circuits of each atomic module, *technology-independent*[11] parameterized delay equations for each atomic module are then derived and listed in Table 1. Preliminary validation of the model against Synopsys timing analyzer in a 0.18 micron[12] CMOS technology found projections to be close (within 2 τ_4). More extensive validation of the model against actual router implementations and other models in different technologies will be beneficial.

4. Pipeline latency results

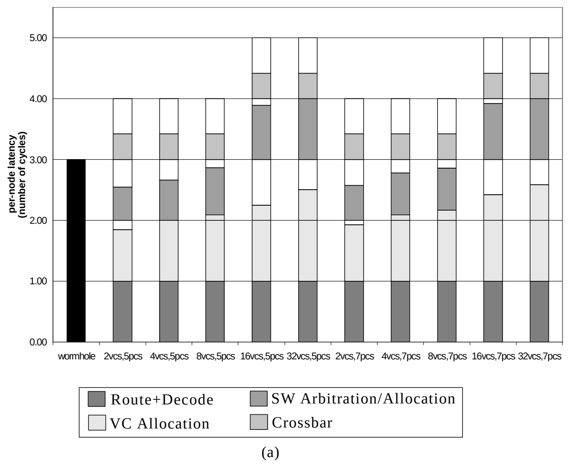
The model of Table 1 enables us to calculate the effect of varying the number of physical and virtual channels on the latency of a pipelined virtual-channel router. Figure 11(a) shows the pipelines proposed by the model for non-speculative virtual-channel routers, with a clock cycle of 20 τ_4. The virtual-channel allocator shown here assumes the most general routing function ($R \rightarrow pv$), and is thus the most complex. For a 2-dimensional virtual-channel router with 5 or 7 physical channels, 4 pipeline stages

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

		p=5; w=32; v=2; clk=20τ_4	
Module	Parametric delay equation (τ)	Model (t_i+h_i) (τ_4)	Synopsys Timing Analyzer (τ_4)
Wormhole router			
Switch arbiter (SB)	$t_{SB}(p) = 21\frac{1}{2}\log_4 p + 14\frac{1}{12}$ $h_{SB}(p) = 9$	9.6	9.9
Crossbar traversal (XB)	$t_{XB}(p, w) = 9\log_8\left(w\left\lfloor\frac{p}{2}\right\rfloor\right) + 6\lceil\log_2 p\rceil + 6$ $h_{XB}(p, w) = 0$	8.4	10.5
Virtual-channel router			
Virtual-channel allocator (VC: $R \rightarrow v$)	$t_{VC:R\rightarrow v}(p, v) = 21\frac{1}{2}\log_4 pv + 14\frac{1}{12}$ $h_{VC:R\rightarrow v}(p, v) = 9$	11.8	11.0
Virtual-channel allocator (VC: $R \rightarrow p$)	$t_{VC:R\rightarrow p}(p, v) = 16\frac{1}{2}\log_4 pv + 16\frac{1}{2}\log_4 v + 20\frac{5}{6}$ $h_{VC:R\rightarrow p}(p, v) = 9$	13.1	13.3
Virtual-channel allocator (VC: $R \rightarrow pv$)	$t_{VC:R\rightarrow pv}(p, v) = 33\log_4 pv + 20\frac{5}{6}$ $h_{VC:R\rightarrow pv}(p, v) = 9$	16.9	15.3
Switch allocator (SL)	$t_{SL}(p, v) = 11\frac{1}{2}\log_4 p + 23\log_4 v + 20\frac{5}{6}$ $h_{SL}(p, v) = 9$	10.9	12.0
(XB)	As above		
Speculative virtual-channel router			
(VC: $R \rightarrow v$)	As above		
(VC: $R \rightarrow p$)	As above	14.6 ($R \rightarrow v$)	16.2 ($R \rightarrow v$)
(VC: $R \rightarrow pv$)	As above		
Speculative switch allocator (SS)	$t_{SS}(p, v) = 18\log_4 p + 23\log_4 v + 24\frac{5}{6}$ $h_{SS}(p, v) = 0$	14.6 ($R \rightarrow p$)	16.2 ($R \rightarrow p$)
Combination of VC and SS (CB)	$t_{CB}(p, v) = 6\frac{1}{2}\log_4 pv + 5\frac{1}{3}$ $h_{CB}(p, v) = 0$	18.3 ($R \rightarrow pv$)	16.8 ($R \rightarrow pv$)
(XB)	As above		

are sufficient for up to 8 virtual channels per physical channel, while a wormhole router fits within a 3-stage pipeline. Thus, for most practical numbers of virtual channels, a

11. Technology independence is inherited through the adoption of the technology-independent τ-model of delay.
12. In a 0.18 micron process, τ_4 = 90 ps. Thus, a 20 τ_4 cycle time is approximately 2 ns, corresponding to a 500 MHz clock.



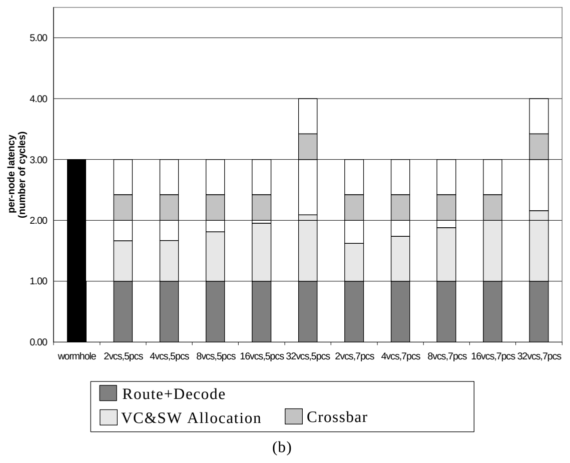


Figure 11. Effect of *p*, the number of physical channels (pcs) and *v*, the number of virtual channels (vcs) on the per-node latency of (a) non-speculative virtual-channel routers and (b) speculative virtual-channel routers. A typical clock cycle of 20 τ_4 was assumed. Each bar illustrates the router pipeline, with the shaded regions corresponding to the fraction of clock cycle time used by each respective atomic module. The 3-stage pipeline of a wormhole router is graphed for reference.

non-speculative virtual-channel router requires just one more pipeline stage than a wormhole router.

Figure 11(b) shows the pipelines derived by the model for *speculative* virtual-channel routers, again assuming a typical clock cycle of 20 τ_4. Here, the routing function ($R \to v$) is assumed. The model indicates that a speculative virtual-channel router with up to 16 virtual channels per physical channel (for 5 and 7 physical channels) fits within a 3-stage pipeline, and thus has the same per-node router latency as a wormhole router.

Figure 12 next shows the effect of different routing functions on the delay of the combined allocation stage in a speculative virtual-channel router. In many configurations, using a less general routing function enables the entire allocator to fit within a single cycle, lowering the per-node latency of a virtual-channel router to that of a wormhole router.

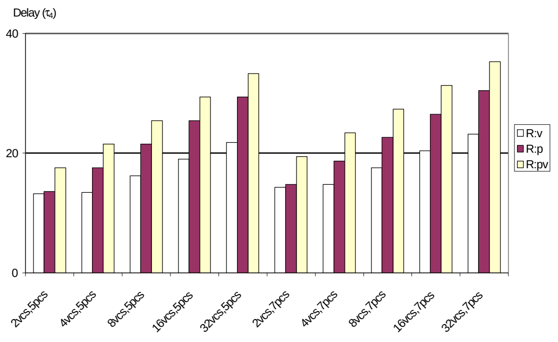


Figure 12. Effect of p, the number of physical channels (pcs) and v, the number of virtual channels (vcs) on the delay of the combined virtual-channel and switch allocation pipeline stage of a speculative virtual-channel router. The effect of different routing functions ($R \to v$, $R \to p$, $R \to pv$) on the complexity and delay of the virtual-channel allocator is also graphed.

5. Simulation results

Based on the pipeline designs prescribed by the model, detailed Verilog code for wormhole, virtual-channel, and speculative virtual-channel routers was written and simulations carried out to determine their latency-throughput characteristics. A typical clock cycle of 20 τ_4 was assumed. The simulator generates uniformly distributed traffic[13] across an 8-by-8 mesh network to random destinations. Each simulation is run for a warm-up phase of 10,000 cycles. Thereafter, 100,000 packets are injected and the simulation is continued till these packets in the sample space have all been received. A constant rate source injects 5-flit packets at a percentage of the capacity of the network and the average latency of the packets is calculated. Latency is calculated from the time when the first flit of the packet is created, to the time 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 the use of buffers, and propagation delay across the channel is assumed to take a single cycle. Since the purpose of our simulations is to explore

13. Uniformly distributed traffic was selected since we are comparing different flow control techniques, which are relatively invariant to traffic patterns, unlike routing strategies.

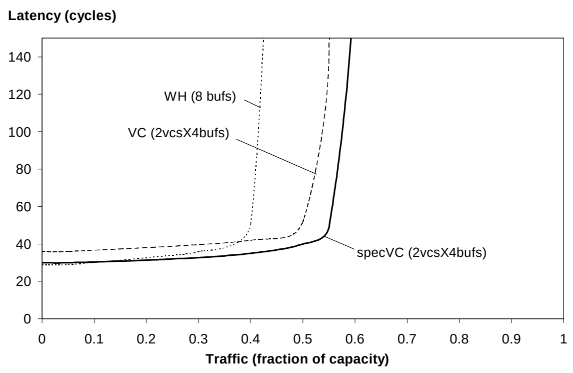


Figure 13. Latency-throughput curves of wormhole, virtual-channel, and speculative virtual-channel routers with 8 buffers per input port. The routers adhere to the proposed pipelined router model.

the performance of flow control strategies, we chose simple dimension-ordered routing[14].

5.1 Comparison of wormhole, non-speculative virtual-channel and speculative virtual-channel routers

Figure 13 shows latency-throughput curves for each of the 3 routers with 8 flit buffers per physical channel. The zero-load latency of the wormhole router (29 cycles) is lower than that of the virtual-channel router (36 cycles), because it has fewer pipeline stages, three vs. four. The speculative virtual-channel router, which also has three pipeline stages has a zero-load latency of 30 cycles[15], comparable to that of a wormhole router.

The figure shows that the throughput of wormhole flow control saturates at 40% of capacity. Virtual-channel flow control with 2 virtual channels extends this saturation point to 50%, and speculative virtual-channel flow control pushes it even further to 55% capacity. This 5% improvement is due to the sensitivity of throughput to pipeline latency, and in particular the latency of the credit path.

Latency-throughput curves for routers with 16 flit buffers per physical channel are shown in Figure 14 (8 flit buffers per VC) and Figure 15 (4 flit buffers per VC). Again, a virtual-channel router has a higher zero-load latency (35 cycles) as compared to that of a wormhole router (29 cycles), due to the additional pipeline stage per hop. The speculative VC router manages to lower the zero-load latency back down to 29 cycles. A virtual-channel router with 2 virtual channels has a throughput of 65% capacity, while a speculative virtual-channel router achieves 70% capacity, a 40% improvement over wormhole flow control (50% capacity). With 4 virtual channels, both virtual-channel routers, speculative and non-speculative, have a throughput of 70% capacity. In this case, there is sufficient buffering to cover the delay of the credit loop, so the reduced pipeline latency of the speculative VC router no longer translates into higher throughput.

14. This is a $R \to p$ routing function, which is the most general possible for deterministic routing.
15. This is a cycle more than that achieved by wormhole flow control because the 4 buffers per virtual channel do not sufficiently cover the credit loop.

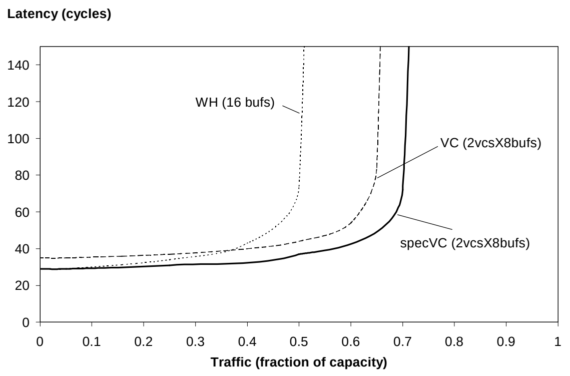


Figure 14. Latency-throughput curves of wormhole, virtual-channel, and speculative virtual-channel routers with 16 buffers per input port. The virtual-channel routers have 2 virtual channels per physical channel. All routers adhere to the proposed pipelined router model.

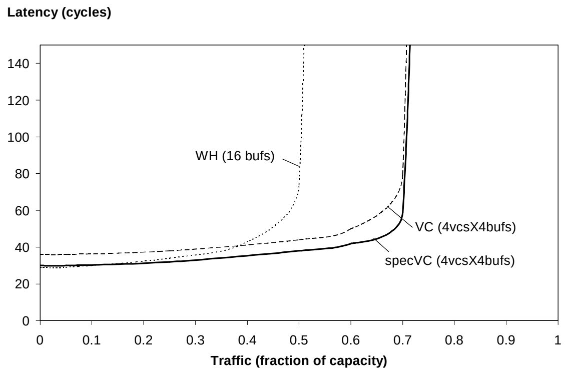
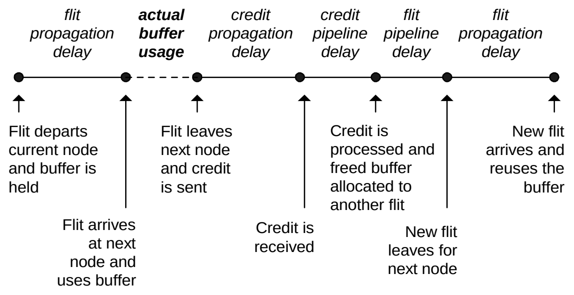


Figure 15. Latency-throughput curves of wormhole, virtual-channel, and speculative virtual-channel routers with 16 buffers per input port. The virtual-channel routers have 4 virtual channels per physical channel. All routers adhere to the proposed pipelined router model.

5.2 Effect of assuming single-cycle router latency vs. multiple-cycle pipelined design

Most published research compares the performance of different router designs assuming single-cycle router



flit propagation delay
actual buffer usage
credit propagation delay
credit pipeline delay
flit pipeline delay
flit propagation delay
Flit departs current node and buffer is held
Flit arrives at next node and uses buffer
Flit leaves next node and credit is sent
Credit is received
Credit is processed and freed buffer allocated to another flit
New flit leaves for next node
New flit arrives and reuses the buffer

Figure 16. Timeline illustrating buffer turnaround time for wormhole and virtual-channel flow control. Buffers are held unnecessarily throughout the propagation and pipeline delays. Thus, the longer the pipeline latency, the longer the buffer turnaround time, and the lower the network throughput.

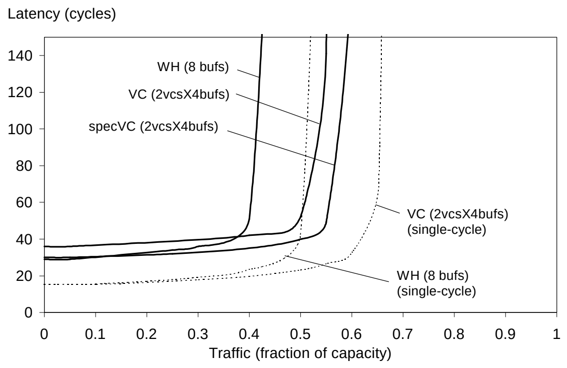


Figure 17. Performance of wormhole, non-speculative and speculative virtual-channel routers, as modelled by the proposed delay model, and as modelled assuming a single-cycle router delay. (8 buffers per input port)

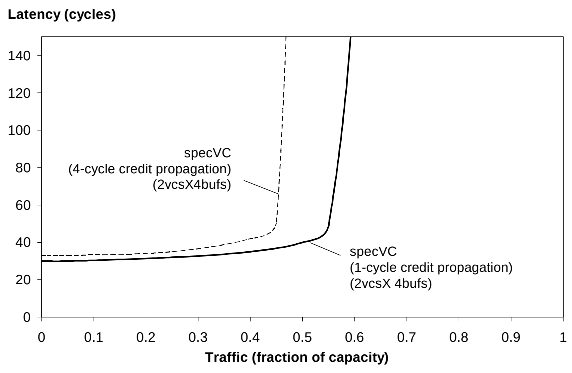


Figure 18. Performance of speculative virtual-channel routers with a 1-cycle credit propagation delay and a 4-cycle credit propagation delay (2 virtual channels with 4 buffers per virtual channel).

latency, without taking into account implementation complexity and cost. This results in inaccuracies in both zero-load latency, since the differences in per-hop latency are not accounted for, and in saturation throughput because it assumes faster buffer turnaround.

Increasing latency, and in particular the latency of the backward credit path reduces the effective amount of buffering in a router by increasing buffer idle time between uses. The timing of this buffer *turnaround* is illustrated in Figure 16. This figure extends the buffer turnaround timeline in [9] to include flit and credit pipeline delays. The timeline in the figure shows how the buffer is held idle for the delay of the *credit loop*, while the credit for the buffer is sent back to the previous node and the next flit is forwarded to the current node. Increased pipeline latency increases the delay of both the credit and flit portions of the credit loop resulting in increased idle time and hence lower throughput.

To quantify this effect, we simulated wormhole, virtual-channel and speculative virtual-channel routers with single-cycle router latency on a cycle accurate "C" simulator and compared the results to the verilog simulations. All other experimental parameters of the "C" simulation are identical to that of the Verilog simulator. Experimental results are shown in Figure 17. With single-cycle router latency, both wormhole and virtual-channel routers have a zero-load latency of 16 cycles, while simulations using the more accurate pipeline models show increased latency for all cases and reflect the larger pipeline delay of the non-speculative VC router.

The figure dramatically illustrates the effect of pipeline latency on throughput. A single-cycle VC router achieves 65% throughput while a realistically pipelined VC router saturates at 50% without speculation and 55% with speculation. This significant difference is due to the effects of buffer turnaround. In a single-cycle router, a credit can be sent and received in 2 cycles, while in our pipelined model, a wormhole and speculative virtual-channel router needs 4 cycles to turnaround credits, and a non-speculative virtual-channel router needs 5 cycles. Thus, throughput is lower in the pipelined model than in one which ignores implementation delay.

To validate that this effect is due to credit latency, we ran simulations of identical speculative virtual-channel routers (2 VCs with 4 buffers each) with a credit propagation latency of 1 cycle, and a credit propagation latency of 4 cycles. In the latter case, 7 cycles are needed to turnaround a credit. The results graphed in Figure 18 clearly show an 18% reduction in throughput from 55% to 45% capacity.

6. Conclusions

Accurate performance models of routers enable architects to tune router parameters for optimum performance before starting detailed design. In this paper we have presented a model of router delay that accurately accounts for pipelining and propose pipelines which are matched to flow control methods. The model uses technology-independent parametric equations for delay that are derived from detailed gate-level designs. Given the type of flow control and key parameters (number of physical channels, number of virtual channels, and phit size), the model gives the overall latency of the router in technology independent units, and the number of pipeline stages as a function of the cycle time.

Motivated by this model, we have introduced a speculative virtual-channel router which optimistically arbitrates for the crossbar switch in parallel with allocating an output virtual channel. Because non-speculative crossbar requests are given priority over speculative requests, the speculation is conservative - i.e., it will never reduce router performance. This speculative architecture largely eliminates the latency penalty of using virtual-channel flow control, reducing the latency of a virtual-channel router to that of a wormhole router. The shorter pipeline of a speculative virtual-channel router also results in increased throughput for small numbers of buffers as it reduces buffer turnaround time.

Using this accurate model we compare the performance of wormhole, virtual-channel, and speculative virtual-channel flow control. Our results show that both virtual-channel routers give a substantial throughput gain over a straight wormhole router, contrary to previously reported results [3].

We compare simulations using our accurate pipelined model with simulations based on a single-cycle router model and find considerable differences between the two models. The single-cycle model greatly underestimates latency by ignoring pipeline delays. It also overestimates throughput by not accounting for buffer turnaround time. These results indicate that accurate pipeline models are needed to get meaningful results from network performance analyses.

Our simulation results highlight the importance of latency in the credit path. While credit latency does not directly impact zero-load latency, it does affect buffer turnaround time and hence has a considerable influence on overall throughput. Our experiments show an 18% reduction in throughput for a speculative virtual-channel router when the credit propagation latency is increased from 1 to 4 cycles.

There are many exciting directions in which the work presented here can be extended. We are currently working on extending our performance model to other flow control methods including flit-reservation flow control [9]. The work can also be extended to consider other topologies and other routing policies, for example, adaptive.

Acknowledgements

We wish to thank the anonymous reviewers for their valuable suggestions and comments. This work was sponsored in part by the Defence Advanced Research Projects Agency (DARPA) under contract number MDA904-98-C-A933.

References

[1] Kevin Bolding et. al., “The Chaos Router Chip: Design and Implementation of an Adaptive Router”, In *Proceedings of IFIP Conference on VLSI*, September 1993.

[2] Andrew A. Chien, “A Cost and Speed Model for k-ary n-cube Wormhole Routers”, In *Proceedings of Hot Interconnects*, Palo Alto, August 1993.

[3] Andrew A. Chien, “A Cost and Speed Model for k-ary n-cube Wormhole Routers”, *IEEE Transactions of Parallel and Distributed Systems*, vol. 9, no. 2, February 1998.

[4] William J. Dally, “Virtual-Channel Flow Control”, IEEE Transactions on Parallel and Distributed Systems, vol. 3, no. 2, pp. 194-205, March 1992.

[5] William J. Dally and J. W. Poulton, “Digital Systems Engineering”, *Cambridge University Press*, 1998.

[6] William J. Dally and Charles Seitz, “The Torus Routing Chip”, *Distributed Computing*, Vol. 1, No. 3, 1986.

[7] Jose Duato and Pedro Lopez, “Performance Evaluation of Adaptive Routing Algorithms for k-ary n-cubes”, In *Proceedings of Parallel Computer Routing and Communication Workshop*, May 1994.

[8] D. R. Miller and W. A. Najjar, “Empirical Evaluation of Deterministic and Adaptive Routing with Constant-Area Routers”, In *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, San Francisco, November 1997.

[9] Li-Shiuan Peh and William J. Dally, “Flit-Reservation Flow Control”, In *Proceedings of 6th International Symposium on High-Performance Computer Architecture*, Toulouse, January 2000.

[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 (editor), MIT Press, 1991.

[11] Ivan Sutherland, Bob Sproull and David Harris, “Logical Effort: Designing Fast CMOS Circuits”, *Morgan Kaufman Publishers*, 1999.