A Study of Wormhole Router Architectures

Elias Dagher Husrev Tolga Ilhan Wael Noureddine Jeffrey Huynh

May 18, 1999

1 Introduction

The main measures of performance of an interconnection network are message latency and throughput. The message latency is composed of inter-router latency and intra-router latency, while the throughput depends on the available resources inside and outside the router. Thus, network performance is directly related to the soundness of the router architecture design choices, which are shaped by various requirements imposed by the network specifications. These design choices are reflected in the specific hardware implementation of an interconnection network's routing method, crossbar arbitration scheme, and flow control policies. To explore the effects of router implementation on performance, we critically analyze the routing, arbitration, and flow control units of the Intel Cavallino, the SGI SPIDER, and the Cray T3E routers. The analysis explores several universal traits of these router units that affect performance, surveys the specific units in each router, and discovers the repercussions that each of the router's units has on intra-router performance. The analysis is concluded by providing some suggestions to enhancing the performance of the router units. In section 2, we discuss in general terms issues related to the implementation of routing, arbitration and flow control methods in interconnection networks routers. Section 3 is devoted to descriptions of the particular implementation choices done in three commercial designs, the Intel Cavallino, the Cray T3E and the SGI SPIDER. In section 4, we comment on these choices and try to assess their adequateness to the stated goals and to the more general measures of performance. Finally, section 5 concludes by a summary of the discussion.

2 Implementation and Performance

The impact on performance of a particular hardware implementation of a function is directly related to the complexity and exactness of the function being performed. Complexity is a measure of the number of inputs, outputs, and steps to perform a function. The exactness of a function states how close to an optimal solution the function is able to achieve. In general, as a function becomes more complex more hardware is needed to implement the function. In turn, more hardware implies a greater latency to complete the function, if the function cannot be parallelized sufficiently. Achieving an exact solution though does not necessarily imply greater complexity or latency, thus making a generalization between exactness and implementation performance is not possible. However, exact methods that must explore a large solution space may indeed be too prohibitive to implement efficiently. As the traits that affect the performance of the routing, arbitration, and flow control units are explored, the complexity and exactness of each implementation will be noted.

2.1 Routing Unit Implementation

The routing unit implementation is inherently affected by the routing method. The total routing latency is the sum of the latencies involved in address decoding, header updating, and routing algorithm execution [4]. This latency characterizes the time needed for the router to resolve the next hop a packet should take. The complexity of the routing algorithm dictates the complexity of the routing unit. Routing methods can either be based on a table lookup or an algorithm. Since table lookups require access to (usually slow) memories holding the tables, minimizing the number of lookups is important. Algorithmic routes can use complex adaptive routing or straight forward deterministic routing.

To illustrate how the routing algorithm affects the performance of the routing implementation, take for example one of the simplest algorithmic routing methods; dimension ordered routing. Dimension ordered routing involves decoding the header flit and making a simple routing decision of whether to progress further in a dimension or move to the next dimension [4]. This routing logic can be implemented using only comparators and logic gates, which alleviates the need of slow memory accesses. Furthermore, dimension ordered routing does not require the use of virtual channels (VC). Thus, the routing logic complexity is significantly reduced because virtual channel management is not needed. Because of the simplicity of dimension ordered routing, the routing unit can achieve low latencies.

In contrast, a routing unit that implements adaptive routing, is significantly more complex than the basic dimension ordered router [9]. In order for an adaptive routing unit to make a routing decision, it needs to employ a non-deterministic routing function, use a selection function, and gather state information on virtual channels. The adaptive routing function can be algorithmic, requiring complex logic. For example, if Duato's algorithm is used, then the routing function logic must be aware of which channel subset the packet is on so as not to violate any channel orderings. The selection function is applied to the routing function results as a final step. The multiple levels of complex logic needed to employ an adaptive routing algorithm suggest that adaptive routing units will have higher latencies. Chien's [4] study on k-ary n-cube router cost and performance concluded that "adaptive routers should not claim advantages in low-load latency, as any apparent advantage is probably eliminated by increases in router setup time;" setup time being the time needed to determine the route taken. In contrast, the latency of routing methods that employ table lookups will have hardware implementation latencies that are dominated by the cycle time of memory accesses. Effectively, the performance of the routing unit is dependent upon the complexity of the logic needed to implement the algorithm. Simple deterministic and algorithmic methods will lend to implementations with shorter latencies. More flexible routing methods that use virtual channels or adaptive routes may require multiple steps, or clock cycles, to produce a route.

The desirable properties for a routing method and the associated routing unit as implemented in a router are speed (low latency), high throughput, network reliability and adaptivity, and flexibility.

2.2 Arbitration Unit Implementation

The arbitration unit within a router is responsible for assigning input buffers to output ports on the router. The main goal of an arbiter is to provide fair access to a service to users in a reliable, low cost, and fast fashion [13]. The primary service that arbiters are used for is the switch crossbar, which provides connectivity between input buffers and output ports. The organization of the input buffers influences the implementation of the arbitration unit. If the input buffers are organized as first-in-first-out (FIFO) queues with one queue per input port, then the arbiter needs only arbitrate between inputs that want to use common output ports. This task can be solved by independent arbiters, one for each output port. However, FIFO buffers are restrictive in that they prevent the optimum use of the crossbar resources [13]. The limitation results because the FIFO is only exposing one request within the queue, when other requests hiding behind the head of the FIFO could be used in making maximum use of the crossbar. If the crossbar is not fully utilized, then the throughput of the router is degraded. Therefore, the use of multi-buffers is beneficial in maximizing the use of the crossbar.

Tamir et al. [13] notes that multi-buffers alone will not maximize the use of the crossbar, but that an effective and efficient arbitration scheme is also needed. The use of multi-buffers transforms the arbitration function into a symmetrical arbitration. The symmetry is that input ports contend for output ports to use, while output ports contended for multiple input ports in an attempt to make full utilization of port resources [13]. Symmetrical arbitration can also be viewed as an allocation problem that employs a two step arbitration; an arbitration between inputs and an arbitration between outputs. However, care must be taken in the implementation of the arbitration unit so as not to bias the two step process, or else the benefits of multi-buffering will be negated by poor arbitration [13]. Furthermore, a two step arbitration can be costly in terms of latency since one clock cycle per step is needed. Therefore the arbitration scheme should adequately perform its task in a timely manner.

A fine balance between "perfect" arbitration and "good" arbitration exists. Perfect arbitration schemes are not feasible to implement in hardware because the scheme either takes too long to run or requires excessive chip area. "Good" arbitration provides equal access to services using a heuristic that is close to optimal. Heuristics can be implemented with simple, fast logic while still achieving acceptable solutions.

An example arbitration scheme, *wave front arbitration*, is given in [13], which models all the cross points of the crossbar as a matrix. The rows of the matrix represent the input buffers and the columns represent the output buffers. The elements in the matrix are marked when an input buffer needs a particular cross point to reach an output. After arbitration, at most one cross point can be assigned to a row or a column. The top left cell has the highest priority over all cells. The arbitration occurs in a greedy fashion between diagonals or waves of the matrix that move from top left to bottom right. After each arbitration cycle the elements of the matrix are rotated to the right, and after n cycles the elements are rotated upwards. Where n is the number of columns in the matrix. This heuristic maintains fairness in the arbitration scheme. The heuristic is rapid, because it is performed in a single step. Ultimately, the arbitration unit must be implemented using a "good" arbiter that is efficiently implementable so as to maximize the crossbar throughput.

2.3 Flow Control Unit Implementation

Wormhole routers are named after the *wormhole flow control* scheme they use [6]. Flow control manages the transmission of phits from the output port of one router to a specific input virtual channel buffer on a downstream router. The flow control policy is implemented by several units that manage state data, virtual channels and input buffers. As a result, wormhole flow control performance depends on the number of virtual channels and input buffer organization.

The number of virtual channels directly affects the complexity of the flow control unit [4]. As more virtual channels are used per port, the flow control becomes more complicated. For example virtual channels can be implemented by dedicating fixed amounts of cycles to each virtual channel on the physical channel. The logic to implement this requires only finite state machines at the transmit and receive ends that channel data through a virtual channel at fixed times. However, this method is wasteful because idle virtual channels will force the physical channel to be idle during their fixed time slots. More virtual channels will result in more wasted physical channel bandwidth. On the other hand, virtual channels can be implemented by dedicating the physical channel on an "as needed" basis. This method has overhead that states the destination virtual channel buffer of each flit. The overhead can be placed out-of-band by having dedicated signal pins, or in-band by attaching flow control information to each flit transferred. Furthermore, virtual channels require virtual channel state to be kept for use the by the routing unit. This state management adds to the overhead information to be transferred between routers and to the chip area to store the state data. Consequently, flow control "complexity increases superlinearly [with] the number of virtual channels" [4] used in the network. Significant justification must exist before adding more virtual channels than are absolutely needed.

Input buffer size affects the bandwidth utilization that can be achieved by a flow control policy. As the input buffer size increases, a higher bandwidth utilization can be achieved because the input buffers have a greater capacity. Larger capacity reduces the frequency of control traffic needed to throttle the input into the buffer in order to prevent overflow. As a result, packets may transfer for longer periods of time without interruption. Increased buffer sizes, though, slow down the responsiveness of the system because fewer occasions occur for the flow control unit to preempt longer data transfers [4]. Wormhole networks have minimal buffer size requirements that must be balanced to achieve high bandwidth and greater system responsiveness. [3] studied the performance improvements derived by varying flit buffer sizes for wormhole routers. The study concluded that 4 to 8 flit buffers for each channel balance cost and performance in wormhole routers that use virtual channels. The study also confirmed that smaller buffers increase the rate of network saturation, while larger buffers retard the rate of network saturation but slowed down network responsiveness.

3 The Intel Cavallino, the SGI SPIDER, and the Cray T3E Router Implementations

In light of the explanations above, we now examine the Intel Cavallino, the Cray T3E, and the SGI SPIDER routers [1][12][10]. One thing to keep in mind is that the target performance in the design of these chips varies with respect to their purpose. For instance, the Cavallino is the backplane router of Teraflops machine which is required to support networks of thousands of nodes (around 9000) in a high-bandwidth low-latency manner (400MB/sec unidirectional data transfer for each direction and less than 50nS node-to-node latency). On the other hand, the Cray T3E Network provides access to a shared distributed memory. It concentrates on sustainable bandwidth and scalability rather than on latency. Note that part of this latency tolerance is made possible by the increased pipelining in the memory system. Finally, the SGI SPIDER is suitable for a variety of high end applications. One important aspect of these applications is interprocessor communication, which is latency sensitive. Consequently, given the motivations behind each network, we examine the resulting design in terms of routing, arbitration and flow control.

3.1 Routing

The Intel Cavallino uses dimension order routing, the order of dimensions being Z-X-Y-Z. Since table based routing increases the latency considerably, the routing decision is made based on a *hop count*, which is decremented appropriately at each hop. The maximum number of hops that are allowed to be taken in a direction before making a turn varies with respect to the direction (8 in Z, 256 in X and 64 in Y). The header flit contains offsets in each dimension, eliminating the need to calculate the offset at each hop. Some of the disadvantages of this routing strategy are that it is hard to change the default configuration in a short time and end nodes need to know the global configuration of the network [1][8, pp. 334-336].

Since the Cray T3E is somewhat latency tolerant and instead targets sustained bandwidth, it uses a table based fully adaptive routing method. Adaptive routing provides a detour around local congestion in a network and in allows sustained network throughput. However, both properties, table based and adaptive, increase latency. Each physical channel in the T3E contains five virtual channel, four of which uses static routing while the last one is reserved for adaptive routing.

The static routing appends destination tags to the packet header rather than appending relative hop count values. These tags the include destination node addresses which are looked-up in a 544 entry table at each hop. This also requires each router chip to have a "who am I" register. If there are more than 544 nodes in a network then 4 processors are mapped on to a single chip. In this case, the last two LSBs of the node address are not used while looking up the node in the table. Consequently, the Cray T3E can support up to 2,176 processors. An entry in a routing table, includes the destination address, the deterministic path to that destination, and adaptive routing information. A special bit value is used to enable routing adaptivity, providing for better fault tolerance. In other words, a message can be routed along a minimal static path set by the source router or along the adaptive virtual channel over any physical link in the minimal path. In the adaptive virtual channel, packets can turn in any direction that moves them closer to their destination. This adaptive routing mechanism is an adaptation of Duato's approach [9], the only difference is that T3E can allow multiple packets to be present in the adaptive channel at the same time. Furthermore, in order to prevent deadlock, the ordering restriction must be preserved upon switching from adaptive channel to a deterministic channel.

The ordering obeyed in deterministic channels between the initial and final hops is direction ordering with the order +X, +Y, +Z, -X, -Y, -Z. The advantage of direction order is fault tolerance. Since the ordering of the directions is fixed, not the ordering of the dimensions, there exist different routes with different corners between two nodes. Therefore, even with a broken intermediate node transmission can continue. This again helps to sustain throughput and improve reliability [12][8, pp. 341-342].

The SGI SPIDER also implements two different routing methods, namely table based routing and vector routing. Vector routing is reserved for network administrative messages while other messages use table-based routing. Routing tables provide a static path between every pair of nodes in the network. These tables are software programmable which is advantageous when a new route is defined or a fault is detected. In other words, scalability is made easier. Even though the table lookups take considerably more time than the processing of source routing messages, an advantage of table based routing is that the end points do not need to have global network knowledge. Each message has a unique destination ID which is used to look up in the table. The table returns an exit port number which is used in the crossbar arbitration. In order to reduce the number of table entries, a 2-level hierarchical routing table is used. The 4 LSBs of the destination ID determine the local address while the 5 MSB bits specify the meta address. If the meta IDs do not match, the meta table is used to route the message to the correct meta domain. Otherwise, the local table is used to route the message to the final destination. While reducing the table size, this method introduces some extra latency. Vector routed messages that contain the network configuration information can be sent while the standard table messages are in flight. Vector routed messages contain the exact route of the packet. Therefore, routing is faster than for the table-based messages [10][8, pp. 345-348].

3.2 Arbitration

The Intel Cavallino has six ports and each of the six ports transfers up to four independent data streams (i.e. four VCs). Therefore the crossbar switch requires a 24-to-1 arbiter. Since the request and the grant signals need to traverse the core of the chip and the clock period is relatively short for this, a two-staged arbitration is performed. In the first stage, input ports are arbitrated for a desired output port (6-to-1 arbitration). In the second stage, the arbiter decides which virtual channel will be used for that output (4-to-1 arbitration). The first arbitration stage is a modified round-robin where once a lane was granted ownership it retained ownership until the message is closed. The second arbitration stage is simple round-robin, taking into account receiver space available [1][2].

As mentioned above the T3E uses direction order, fully adaptive routing. In order to decrease the latency, it uses a short-circuit technique for packets that are not turning on that router. In other words, each interface in the router contains a short-circuit path so that packets that do not change direction on that router can exit the router without entering the crossbar switch. If there is contention on one of the interfaces, the arbiter can force the other interfaces to give up their short-circuit paths and use the crossbar.

There are two stages of arbitration. The first stage is among the input VCs of a given interface to decide which one gets to arbitrate for the output VCs. The second stage is the arbitration among the output VCs. In the first stage, each interface arbitrates among its VCs in a round-robin fashion. The winner holds its winner status until its message completes. However, if the winner loses in the arbitration for the output channel or its tail and header are separated then it is forced to release its status. The winner of the first stage can make requests to more than one output arbiter. One of these request is for one of the deterministic VCs. Another request is made for the adaptive channel in the highest direction order if the adaptivity is allowed by the packet. If both the deterministic and the adaptive requests are available, then the adaptive one is chosen unless they use the same physical channel. In that case, the deterministic one is preferred. The second stage arbitration is for the physical channel among the output VCs and is also performed in round-robin fashion. However, the adaptive channel has the lowest priority in this case.

The SGI SPIDER chip uses a Wavefront Arbiter as described above. To maximize the bandwidth through the crossbar, each virtual channel buffer is organized in a linked-list form. One linked-list is designated for each output channel. This way a blocked message demanding one output port will not keep other messages from progressing if they are requesting other output ports.

Fairness and starvation prevention is achieved by prioritizing the packets. Each packet header has an age field which can be interpreted as the packet priority. The message will age at a programmable rate while the packet is waiting in a router and the older the packet gets the higher its priority becomes. Also, messages considered high priority can be injected into the network by setting the age field at a high age. However, if there is no contention for the output port, there is no need to waste any time on the priority decision. In this case the SGI SPIDER uses fixed priority to decrease the latency (e.g. input port 1 always wins over input port 2 and so on). Furthermore, the SGI SPIDER provides a programmable option which prevents other message packets from cutting through the message in progress. This can decrease the latency of high priority messages, while causing low priority messages to wait longer.

3.3 Flow Control

Intel Cavallino has 3 control bits parallel to the data lane which provides information about data framing, lane status, and the receiver-queue status. The buffers at each lane can signal the sender of the data to stop stop transmitting when they are almost full. This can be accomplished by using *Almost Full* bits.

The T3E uses *credit-based* flow control over each virtual channel. Credits are piggy-backed on data packets. Each deterministic virtual channel has a 12-flit deep buffer. If the channel is adaptive, it uses a larger buffer (22 flits). There are different types of packets such as a single word to 8-word read or write request, 8-word message packets, atomic memory operation packets, and system configuration packets. It uses 4 deterministic and one adaptive virtual channel for each physical channel. The virtual channels spread the traffic evenly across the channels which, in turn, increases the sustained throughput. Since each direction is independent of each other, VC assignment is optimized for the single ring on that direction. In order to prevent deadlock, cyclic dependencies in VC buffers are removed by having a dateline node. Except for the adaptive channel, deterministic channels are grouped in pairs and upon crossing of the dateline on a VC, a packet is switched to the other VC in the pair. This way cyclic dependencies are eliminated. All other nodes that do not cross the dateline are unconstrained and VC usage is balanced by choosing virtual channels among these unconstrained routes.

The SGI SPIDER manages flow control with flits, referred to as micropackets by the designers, of 17 bytes in length. The reason for the small flit size is that most of the transactions are one or two flits long. Virtual channel buffer management is handled by credit based protocol. Data is protected on a single link instead of end to end. If an error occurs, go-back-n sliding window protocol is used for recovery. Also the SPIDER router can negotiate to use some portion of the channel (10 bit, 20 bit or wider). The data rate is a function of the negotiated width while the interface to the chip stays the same.

4 Critical Analysis

In this section, we analyze the design of the three wormhole routers presented in the previous section. In turn, we examine the routing, arbitration and flow control units of the Intel Cavallino, the Cray T3E router, and the SGI SPIDER keeping in mind the performance tradeoffs discussed in section 2.

4.1 Routing

As described above, the Cavallino has several features geared towards achieving low latency through routers. Thus, it uses the simple and fast dimension-order algorithmic routing. This translates into small overhead for messages and therefore lower latency than the T3E. The use of offsets allows fast checking at intermediate routers, however, it requires header updates to be performed at each hop (these can be performed on the fly as the flit is being switched through the router though). In addition, source nodes need to have some knowledge of the network topology in order to set the initial offset values. Finally, by appropriately setting the source nodes, it is possible to route around downed links and faults. However, this seems to require signifcant human intervention. In fact, the hot-swapping capability refers to built-in hardware that allowed disabling one node, physically removing and replacing that node, and restarting it without having to bring down the entire backplane. The hardware supports the flushing of any open messages coming from or going to that node and disabling additional message routing to that node until the new node is available. Deadlock avoidance is taken care of by the routing software, although the internal arbitration policies guaranteed forward progress [2]. Finally, the head of line blocking is alleviated by the supply of four VCs per physical channel and.

In the Cray T3E, as described above, routing is fully adaptive through the use of an extra adaptive VC in addition to 4 other VCs. Routing is destination address based and is done by accessing a routing table. This translates into a relatively large latency overhead. Again, the designers are not focusing on latency but on bandwidth. The table routing is a solution to make the routing fault-tolerant. However, this means that faulty links must be detected and all the tables (at each router) reconfigured. A better solution for lower latency is to use algorithmic routing. However, an algorithmic scheme is used by the router to avoid deadlocking adaptive routes. That is, the router uses direction-ordered routing, with the order +X, +Y, +Z, -X, -Y, -Z, to avoid turn cycles, and a dateline scheme to avoid torus cycles. Faults can be avoided by non-minimal routing around the fault, with the help of the adaptive VC. The adaptive VC network is not deadlock-free, but if deadlock happens, the messages can switch to the normal network, which is deadlock-free. The advantages gained are a better load balance and thus a higher aggregate throughput, fault tolerance and flexibility. The adaptivity available comes at the expense of speed (higher latency). The associated steps of header decode, table lookup and deadlock prevention computations are in the critical path of the router.

The SPIDER also uses programmable, distributed tables for routing. Several properties result in a routing overhead that is much reduced compared to the T3E router. First, the table size is reduced by the use of a hierarchical routing scheme. Second, the table access is pipelined across routers, where a router uses the routing information obtained in the previous router and provides the routing information to be used by the following router on the way to the destination. In fact, although it is restrictive, the hierarchical arrangement is especially suitable for the multidimensional hypercube topology the chip is designed for. An algorithmic may not work as well, simply because the topology is not as simple as a 3D cube. However, an adaptive routing scheme would have helped load balance the traffic, a feature that is not possible with the SPIDER.

The Intel Cavallino seems to achieve its goals. However, it might be the case that the requirements are not very tight (at least not for today's technology). It is safe to predict that much lower per hop latency is feasible, given the simplicity of the design, in addition to higher wire signaling rates. Such improvements made available by advances in technology can in fact be used to improve all three routers. Although the stated goals of the T3E do not include low latency, the performance figures provided in [12] are several times larger than those of the SGI SPIDER network for the same size. It seems that the routing in the T3E would gain from implementing some of the ideas that are used in the SPIDER, resulting in a reduced latency. However, such changes would need to preserve the routing flexibility of the T3E. The most promising candidate for that could be the use of inter-router pipelining of operations.

4.2 Arbitration

The use of arbitrators in the three routers studied here has two main goals. The first goal, which is the primary one, is to select the VCs that are going to be acquiring the output channels in the next cycle. The other goal is the reduction of the interior crossbar switch size, by performing arbitration at the input ports.

The Intel Cavallino router uses two levels of arbitration, performed independently, as described in the previous section. This separate allocation has limitations and results in throughput loss at the router crossbar switch [5]. The Cavallino designers justify this choice by citing clock cycle constraints on the logic speed. It appeared impossible to perform a 24-to-1 arbitration in one clock cycle because of the distributed nature of the implementation, some of the tightest speed paths being in the arbiters. Once they chose two stages, the arbitration policy for each stage was made relatively simple. If one could do the arbitration in one stage instead of two, it might be possible to reduce the latency by one clock. However, the arbitration policy would have to be reworked to guarantee forward progress and would be substantially more complex. Also, one has to reexamine the entire internal pipeline and repartition it in light of the reduced latency [2]. This limitation is probably more relaxed today, given the improvements in chip speeds, and better arbitration schemes are now possible. Such schemes would result in increased network throughput and reduced latency.

The T3E router uses round-robin arbitration, which is not completely fair. However, due to the complexity and sizes of other units in the chips (i.e. the buffers and the routing unit) there is probably not much room for more complex arbiters. To have completely fair arbitration, one can use matrix arbiter, which is based on the last-used-last-served rule, or the queuing arbiter, which is based on the first-come-first-served rule. Both of these types of arbiters are more complex than the round-robin arbiter, which may be just a simple shift-register or counter, but provide more fairness.

In the SPIDER router, arbitration is done via a single arbiter. Fairness is achieved using aging bits and thus is an improvement over the round-robin scheme of the T3E router. However, the benefits of such a mechanism are not directly obvious, given the significant added complexity. Moreover, if it performs as intended, the aging mechanism results in a decreased variance in latency at the expense of an increased average latency, which is not always desirable. In fact, this mechanism is bypassed in times of low load (no contention). Note that some latency can be hidden in the SPIDER as the arbitration and the routing are done in parallel. Therefore the designer can afford some extra complexity in the arbitration unit. Another design idea in the SPIDER is the arrangement of flits in per output port linked lists at the input ports, which is aimed as increasing the throughput. Such an organization has been shown to result in excellent throughput results when combined with a good arbitration scheme [11].

4.3 Flow Control

As indicated earlier, the Cavallino does not use credit-based flow control. Instead it uses warning signals that are sent to the upstream router on separate bit lines, when the buffers are “almost full”. The buffer occupancy threshold that the warning corresponds to is programmable, to allow for different input link lengths. The reason this simpler scheme is chosen over credit base signaling is, according to the designers, improved reliability and recovery. At the signaling speed used and the state of the technology, credit-based approach would increase the number of signaling bits required and increase the probability of error. Another simplification is added by the use of equal wire signaling and internal clock frequencies. Today, the wire signaling speeds that are possible are quite higher than the internal chip clocks and (de)serialization delay is usually incurred at the boundaries.

The T3E router uses 5 input VCs (with 1 adaptive VC) and 5 output VCs. Each adaptive VC buffer holds 22 flits, while the other virtual channels use 12 flits. These buffer sizes are larger than needed, since the message length is only 1 to 10 flits. Also, since the round trip time across a link is about two flits, only two flits are necessary for effective wormhole flow-control. The flow control is thus more like virtual cut through than wormhole. Although they are very helpful when there are high network loads, much of the buffering resources would be unused most of the time. Also, deep buffers tend to reduce the clock frequency because latency is increased [8]. However, large buffer sizes could be justified here since the emphasis is on sustained bandwidth, not latency. As long as there is enough chip space for buffers, large buffer sizes would improve bandwidth. So when a message is blocked, it occupies at most one single node, freeing up previous nodes in the wormhole path [12].

The T3E’s way of sending credits (piggy-backed on flits) demonstrates a good way to maximize the use of bandwidth. The other way of sending credits is to have designated signal lines, but this would reduce the bandwidth since the signal lines would be used only once a cycle, while data can be sent 5 times a cycle. The advantage would be its simplicity, which translates into less thinking time. Finally, the T3E uses a fast lane for packets that are going through the router on the same dimension by allowing such packets to take a "short circuit" between the two ports of that dimension. This enhancement helps reduce latency by eliminating the need to arbitrate for and go through the crossbar.

The SPIDER uses 256 byte buffers for each virtual channel. This size is again larger than necessary for basic wormhole flow control, but if message sizes are large, this would help relieve congestion in case of heavy traffic as in the case of the T3E router. However, this convenience again comes at the expense of chip area. The SPIDER's flow control is credit-based and maximizes the use of available bandwidth as in the T3E router. Also, the buffer organization into linked lists helps prevent head of line blocking, but it increases the complexity of the buffers. In addition, flits of different messages may share the same VC or buffer. For small messages, this may mean several messages can sit in the same VC and the same time. This is a good idea for highly congested areas, but if traffic is light, these resources may not be useful and may increase the latency, so there is a bypass channel which allow messages to bypass the buffers when the buffers are empty. Finally, the negotiated channel width between two adjacent routers may introduce additional complexity that could increase the transmission latency. It is not obvious where such a capability would be useful in a homogeneous network, which is the common architecture today.

5 Summary

In this paper, we presented and analyzed three commercial routers that implement wormhole flow control. These routers are used in high performance parallel computer interconnection networks, supporting link speeds varying from 400MB/sec to 1 GB/sec. We focused on three main functional units of the routers, namely the routing, arbitration and flow control units, chosen for their direct impact on perceived performance.

All three routers seem to achieve their stated goals, which differ in their requirements. These differences made it possible to isolate the different optimizations introduced by the designers in order to achieve the respective goals. We explored such aspects in our discussion of each router's architecture. We highlighted some good ideas that could be used in any router with similar requirements, or could be implemented in the other routers studied. For example, Cavallino's two-stage arbitration could be replaced by SPIDER's wave-front arbitration to shorten the latency and increase the achieved throughput. Other interesting ideas include pipelining the operations across successive routers to reduce latency as in the SPIDER, the "short-circuit" path used by the T3E router and the "almost-full" flow control technique used by the Cavallino.

An "ideal" router would include these ideas for reducing latency. In addition, it would use table and adaptive routing for flexibility, aging bits for fair arbitration, linked list VC buffers for better throughput, and piggy-backed credits for better usage of bandwidth.

References

[1] Carbonaro, J. and F. Verhoorn, "Cavallino: The Teraflops Router and NIC", *Intel Corp. Technical Report* 1996:157-160.

[2] Carbonaro, J. , "Some Cavallino Design Decisions", *Email to the authors*, May 1999.

[3] Chang, Chin-Yuan, Ting-Wei Hou, and Ce-Kuan Shieh. "Performance Improvement of Wormhole Router for Multicomputer Systems." *Proceeding TENCON 93. 1993 IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering* 1 (1993): 254-257.

[4] Chien, A.A. "A Cost and Speed Model for k-Ary n-Cube Wormhole Routers." *IEEE Transactions on Parallel and Distributed Systems* 9.2 (1998): 150-162.

[5] Dagher, E. et al. , "Homework 3: Fun With Arbitrators", *Stanford University*, EE482, Spring 1999.

[6] Dally, W. J. and C. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks", *IEEE Transactions on Computers*, May 1987, pp. 547-553.

[7] Dally, W. J., "Virtual Channel Flow Control", *IEEE Proc. 17th In. Symp on Computer Architecture*, May 1990, pp. 60-68.

[8] Duato, J. et al, "Interconnection Networks: an Engineering Approach", IEEE Computer Society, 1997.

[9] Duato, J, et al. "A High Performance Router Architecture for Interconnection Networks." *Proceeding of the 1996 International Conference on Parallel Processing* 1 (1996): 61-68.

[10] Galles, M., "Scalable Pipelined Interconnect for Distributed Endpoint Routing: The SGI SPIDER Chip", *Silicon Graphics Computer Systems Technical Report* (1996): 141-146.

[11] McKeown,N., Anatharan V. and Walrand J., "Achieving 100% Throughput in an Input-Queued Switch", *IEEE Infocom 1996*, March 1996, San Francisco.

[12] Scott, S. and G. Thorson, "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus", *Cray Research Technical Report* (1996): 147-156.

[13] Tamir, Yuval and Hsin-Chou Chi. "Symmetric Crossbar Arbiters for VLSI Communication Switches." *IEEE Transactions on Parallel and Distributed Systems* 4.1 (1993): 13-27.