

# Separated High-bandwidth and Low-latency Communication in the Cluster Interconnect Clint

Hans Eberle, Nils Gura  
Sun Microsystems Laboratories  
{hans.eberle,nils.gura}@sun.com

## Abstract

An interconnect for a high-performance cluster has to be optimized in respect to both high throughput and low latency. To avoid the tradeoff between throughput and latency, the cluster interconnect Clint<sup>1</sup> has a segregated architecture that provides two physically separate transmission channels: A *bulk channel* optimized for high-bandwidth traffic and a *quick channel* optimized for low-latency traffic. Different scheduling strategies are applied. The bulk channel uses a scheduler that globally allocates time slots on the transmission paths before packets are sent off. This way collisions as well as blockages are avoided. In contrast, the quick channel takes a best-effort approach by sending packets whenever they are available thereby risking collisions and retransmissions.

Simulation results clearly show the performance advantages of the segregated architecture. The carefully scheduled bulk channel can be loaded nearly to its full capacity without exhibiting head-of-line blocking that limits many networks while the quick channel provides low-latency communication even in the presence of high-bandwidth traffic.

## 1. Introduction

By clustering systems, *availability* is improved and *computing performance* is increased. When compared with other hardware-based approaches such as fault-tolerant systems and dedicated parallel systems, clusters offer a cost advantage since they capitalize on the economies of scale by using relatively inexpensive building blocks.

Here, we are interested in high-performance clusters. Thanks to publicly available software libraries such as implementations of PVM [20] and MPI [17], developing parallel applications for clustered systems has be-

come increasingly popular. Typically, these applications require tight coupling of the processing nodes and, with it, an interconnect that provides high bandwidth as well as low latency.

Clint is targeted specifically at small- to medium-sized clusters that offer a low-cost alternative to SMP systems. Given this design point, we adopted a topology that relies on a single central switch. The limited size of such a cluster offers ample opportunities to optimize cost and performance.

Applications of the Clint architecture are not restricted to high-performance clusters. For example, the segregated architecture and buffer-free switch design are applicable to backplane interconnects used in network routers and switches [16] as well as in server machines and storage devices.

Several cluster interconnects tailored for low-latency communication have been developed. Examples are HIPPI [7], ServerNet [8], SCI [5] and Myrinet [1]. Unfortunately, these interconnects can provide low latency only as long as they are lightly loaded. The reason is that there is a tradeoff between high throughput and low latency. Clint removes this tradeoff by segregating the interconnect into two physically separate channels: A *bulk channel* optimized for the transmission of large packets at high bandwidth and a *quick channel* optimized for the transmission of short packets at low latency. The segregated architecture of Clint is motivated by the observation that network traffic typically falls into two categories: There is high-bandwidth traffic mainly consisting of large packets and there is low-latency traffic mainly consisting of small packets. Such a bimodal packet distribution has, for example, been reported in [11].

We have structured the paper as follows. Section 2 gives an overview of the Clint architecture. Section 3 describes the components of Clint in detail. Section 4 contains simulation results. Section 5 contrasts Clint

---

<sup>1</sup> Clint is the code name of a Sun Microsystems Laboratories internal project.

with related work. Finally, Section 6 gives the conclusions.

## 2. Overview

The Clint interconnect uses physically separate networks to implement the bulk channel and the quick channel. That is, the channels use separate links as well as separate switches.

Given the different optimization criteria, the methods for scheduling the channels look quite different. The bulk channel applies a relatively compute-intensive algorithm that avoids conflicts in the switch and that optimizes the aggregate throughput of the switch. In contrast, the quick channel uses a best-effort approach in that nodes send off packets without any coordination. As a result, packets can collide. When this happens, one packet wins and the other packets lose. The winning packet is forwarded and the losing packets are dropped and retransmitted at a later time – retransmissions are handled in hardware.

The Clint network forwards packets with fixed delays. This property simplifies packet scheduling and error detection. More specifically, it allows for a pipelined operation of the bulk channel under control of a global schedule. Further, it simplifies error detection since a simple request-acknowledge protocol can be used: If an acknowledgment packet has not been received a fixed amount of time after the request packet was sent, a transmission error occurred. That is, while similar protocols have to cope with variable, possibly unbound delays and variable numbers of outstanding packets, the protocol for Clint can rely on bounded delays and a fixed number of outstanding packets.

We have implemented a prototype system with the following characteristics. Each channel uses a single switch with 16 ports. Links are serial copper cables with a length of up to 5 m. The bulk channel has a full-duplex bandwidth of 2+2 Gbit/s and the quick channel has a full-duplex bandwidth of 0.53+0.53 Gbit/s. With it, the bulk switch has an aggregate bandwidth of 32 Gbit/s and the quick switch has an aggregate bandwidth of 8.45 Gbit/s. The physical layer uses 8B/10B encoding; thus, the given data rates have to be increased by 25% to obtain the link rates.

An implementation of the segregated architecture does not necessarily require separate physical channels. An alternative implementation could use separate virtual channels mapped onto a single shared physical channel. We did not choose this approach since we wanted to avoid the temptation to share resources and,

as a result, compromise on the objectives set forth for each channel.

## 3. Architecture

This section discusses the architecture of Clint in detail. We use the following notation. The node that initiates a transfer is called the *initiator* and the node addressed by the initiator is called the *target*. The packet sent from the initiator to the target is called a *request packet*; the response sent back from the target to the initiator is called an *acknowledgment packet*. Clint implements a *push model* as data is always pushed from the initiator to the target: The request packet contains an *active message* [14], and the acknowledgment packet contains a transmission report.

### 3.4 Network Interface Card

The network interface card (NIC) is based on Active Messages 2.0 [14] and the Virtual Network abstraction [2],[15]. This abstraction virtualizes the access points of the network in the form of *endpoints*. A collection of endpoints forms a virtual network with a unique protection domain. Messages are exchanged between endpoints, and traffic in one virtual network is not visible to other virtual networks.

There are two types of active messages: A bulk message sent over the bulk channel and a quick message sent over the quick channel. The message formats are shown in Figure 1. The fields identify the *initiator* and *target*, the *handler* to be called upon receipt by the target, and the *arguments* of the handler. The bulk message contains an additional 2 kByte payload. Not shown are the packet fields added by the link layer. They include a packet type, a sequence number, and a CRC. The chosen packet sizes are based on studies such as [11] and could be easily changed if needed.

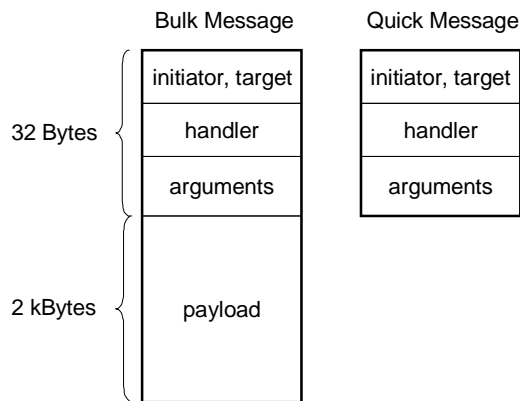


Figure 1: Message Formats.

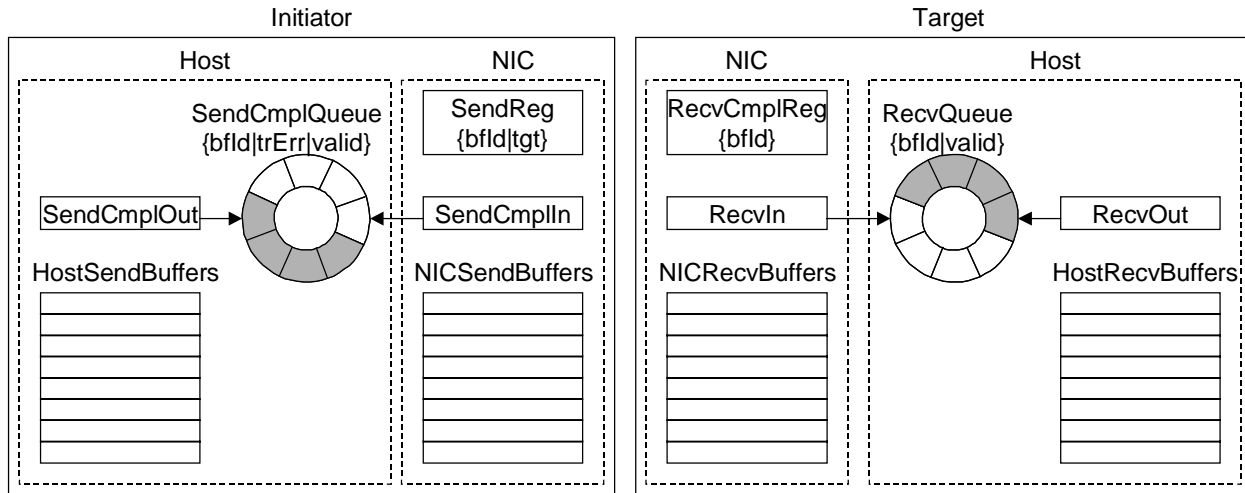


Figure 2: Data Structures for Sending and Receiving Messages.

Figure 2 shows the data structures of an endpoint used for transmitting messages. The data structures are identical for both the bulk channel and the quick channel. They are kept in main memory on the host side and in local memory on the NIC side. Data is exchanged between the host and the NIC via the PCI bus. Since the overhead of PCI bus accesses initiated by the host is considerably higher than for accesses by the NIC, data structures are organized such that the host has to access the NIC as little as possible<sup>2</sup>.

A two-way handshaking protocol manages the buffer pools for sending and receiving messages.

The steps for sending a message are:

1. The host gets a free buffer.
2. The host writes the message into *HostSendBuffers*.
3. The host notifies the NIC of the ready message by writing the corresponding buffer index *bfld* and the target *tgt* of the message into *SendReg*.
4. The NIC transfers the message from *HostSendBuffers* to *NICSendBuffers*.
5. The NIC requests a switch slot and, once the request is granted, sends the message in a request packet.
6. The NIC waits for the acknowledgement packet and, once it is received, notifies the host by enqueueing an entry into *SendCmplQueue*. The entry contains the buffer index *bfld*, possible transmission errors *trErr*, and a *valid* bit.
7. The host dequeues the entry from *SendCmplQueue* and adds the buffer to a free list.

The steps for receiving a message are:

1. The NIC writes the received message into *NICRecvBuffers*.
2. The NIC gets a free buffer.
3. The NIC transfers the message from *NICRecvBuffers* to *HostRecvBuffers*.
4. The NIC notifies the host of the received message by enqueueing an entry containing the buffer index *bfld* and a *valid* bit into *RecvQueue*.
5. The host dequeues the entry from *RecvQueue*.
6. The host notifies the NIC of the completed message by writing the buffer index into *RecvCmplReg*.
7. The NIC dequeues the entry from *RecvCmplReg* and adds the buffer to a free list.

The NIC needs to consume *SendReg* and *RecvCmplReg* faster than the host can write these registers. These registers are, therefore, implemented as the input port of a FIFO structure.

*SendCmplQueue* and *RecvQueue* are implemented as ring buffers accessed by the in-pointers *SendCmplIn* and *RecvIn*, and out-pointers *SendCmplOut* and *RecvOut*, respectively. Since the queues contain as many entries as there are entries in *SendBuffers* and *RecvBuffers*, respectively, they cannot overflow. For the same reason, the queues can be used to maintain the free buffers, thus no separate free lists are required. Referring to Figure 2, the non-shaded entries of *SendCmplQueue* and *RecvQueue* contain indices *bfld* that point to empty buffers. Thus, a free list can be maintained by adding another pointer to these queues. Since we want to keep the free lists in memories local to the host or NIC, respectively, we only used this optimization on the initiator side. On the target side, we maintain a separate free list on the NIC.

The two channels use different prefetching strategies to fill *NICSendBuffers*. The quick channel prefetches

<sup>2</sup> For the platform we used, there is a significant overhead for PCI transactions initiated by the host as only 32-bit bus transfers are supported and transfers require an address translation executed by an IOMMU.

packets in the order their buffer indexes were written into *SendReg*. The bulk channel uses a more sophisticated method in that it tries to fill the buffers with messages destined for different targets. For this reason, *SendReg* also specifies the target so that the NIC has the opportunity to decide in which order to prefetch messages without having to fetch the message descriptor from host memory. By having messages available in *NICSendBuffers* destined for different targets the bulk scheduler is presented with more options and head-of-line blocking is avoided [9].

Though future work will have to explore and analyze the different techniques for prefetching messages, we want to outline a possible method. For each target, the NIC maintains a count of messages that are buffered on the NIC. There is a maximum count that limits the number of messages that can be buffered for each target. Once the maximum count for a target has been reached, the target is no longer considered when making prefetching decisions. This method avoids HOL blocking in that a target that causes blockages cannot consume space in *NICSendBuffers* that could be used for forwarding messages to targets that are not blocking.

It is important to note that HOL blocking can occur not only in the network switches but also in various other places. HOL blocking can potentially be caused in any place that serializes the transfer of data. As we have just shown, possible places other than the switch are the host memory and the NIC. Careful scheduling of the data transfers out of the host memory and out of the NIC becomes even more critical as the bandwidth of the network increases relative to the bandwidths of the main memory and IO subsystem.

### 3.2 Switch Organization

Figure 3 illustrates the different organizations of the bulk switch and the quick switch. In this example, 2x2 switches connect two nodes. Each node consists of a bulk initiator *bini*, a bulk target *btgt*, a quick initiator *qini*, and a quick target *qtgt*.

Most notably, the switches do not contain any buffer memory - the buffer memories shown are located in the nodes. The bulk channel uses an analog switch that corresponds to a flow-through crossbar switch whereby packets are forwarded without being interpreted. This is possible since all signaling for the bulk channel is done via the quick channel. Ports as well as internal data paths are serial. The quick channel is realized as a pipelined crossbar switch. While its ports are serial, the internal data paths are byte-parallel and split into six pipeline stages to allow for a data rate suitable for handling packet collisions and processing packets.

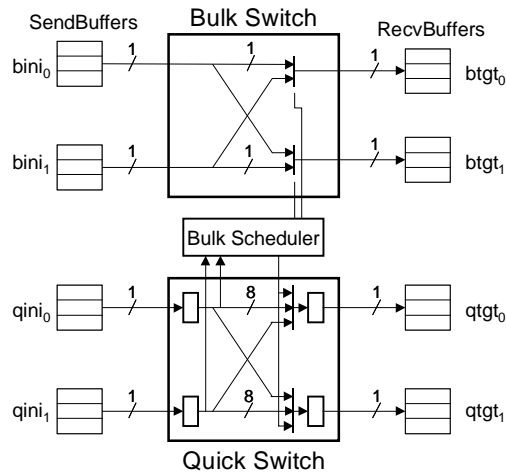


Figure 3: Organization of the Switch.

The prototype switches have the following latencies: 1.5 ns for the analog switch of the bulk channel and 90 ns for the pipelined switch of the quick channel. Of more interest are the round-trip times: They are 20  $\mu$ s for the bulk channel and 1  $\mu$ s for the quick channel. The calculated round-trip times start when the beginning of a request packet leaves the node and stop when the end of the corresponding acknowledgment packet has been received.

A central scheduler connected to the quick channel is used to schedule the bulk channel. By informing the bulk scheduler of all packets waiting for transmission in the send buffers located on the nodes the switch can be optimally scheduled. In particular, HOL blocking [9] can be avoided since, for each initiator, the bulk scheduler can choose from packets with different destinations.

For a pair consisting of an initiator and a target, packets are transferred in-order. The ordering relative to packets delivered to other targets or sent by other initiators is not specified.

The two channels transport the packet types shown in Table 1. The table lists the packet types, their sizes (including header and checksum) and channel assignments. It is only the bulk request packets that use the

Type	Function	Channel	Size [byte]
breq	bulk request packet	bulk	2086
back	bulk acknowledgment packet	quick	4
qreq	quick request packet	quick	36
qack	quick acknowledgment packet	quick	4
cfg	configuration packet	quick	19
gnt	grant packet	quick	4

Table 1: Packet Types.

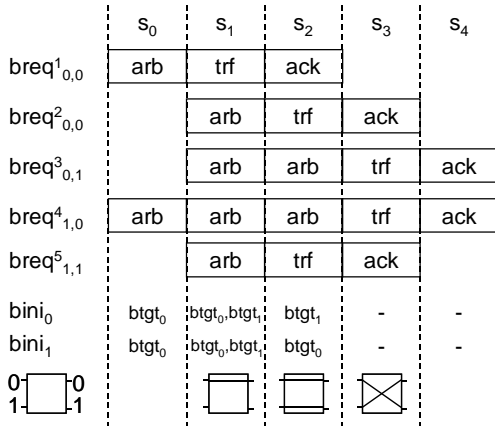


Figure 4: Pipelined Transmission of Bulk Packets.

bulk channel while all other types of packets use the quick channel.

### 3.3 Bulk Channel Pipeline

The bulk channel is operated as a pipelined data path. Packet transmission is split into three stages each having the length of one bulk slot which corresponds to the time it takes to transfer a bulk request packet:

- B1. **arb**: During the *arbitration stage* a connection of the bulk switch is allocated for forwarding the request packet.
- B2. **trf**: During the *transfer stage* the request packet is transferred from a send buffer of the initiator to a receive buffer of the target.
- B3. **ack**: During the *acknowledgment stage* the target returns an acknowledgment packet to the initiator.

Only the transfer stage uses the bulk channel. Both the arbitration stage and the acknowledgment stage use the quick channel. Possible conflicts during the transfer stage are resolved in the arbitration stage. Figure 4 illustrates the operation of the bulk channel pipeline. The transmission of five packets  $breq^n_{i,t}$  is depicted, where  $n$  is a number identifying the packet,  $i$  stands for the initiator and  $t$  for the target. In addition to the transmitted packets, the figure shows the targets requested by the initiators and the settings of the switch. Note that the initiator generates a request for every target for which it has a full send buffer and that, as a result, the initiator can request more than one target in a given arbitration cycle.

In slot  $s_0$ , both  $bini_0$  and  $bini_1$  request connections with  $btgt_0$ . The request of  $bini_0$  is granted and the request packet  $breq^1_{0,0}$  is transferred in slot  $s_1$ . In slot  $s_1$ , both  $bini_0$  and  $bini_1$  request connections with  $btgt_0$  and  $btgt_1$ .  $bini_0$  is granted the connection with  $btgt_0$ , and  $bini_1$  is granted the connection with  $btgt_1$ . The corresponding transfers of packets  $breq^2_{0,0}$  and  $breq^5_{1,1}$  take

place in slot  $s_2$ . The remaining two requests are granted in slot  $s_3$  and the corresponding packets  $breq^3_{0,1}$  and  $breq^4_{1,0}$  are transferred in slot  $s_3$ .

A central arbiter is used to schedule the bulk channel. The arbiter calculates the schedule for the subsequent transfer stage in the arbitration stage. An arbitration cycle starts with each node sending a *configuration packet* to the arbiter. This packet combines information supplied by both the initiator and the target residing on a node. It contains a request vector that identifies the targets for which the initiator has data packets. Also contained in the configuration packet is an enable vector that names the initiators from which the target accepts packets. An initiator is disabled, for example, if it is suspected to be malfunctioning. Separate enable vectors are provided for the quick and the bulk channel. The arbiter can now calculate a conflict-free schedule. The schedule is communicated to the initiators with the help of *grant packets*. Each initiator receives a grant packet that reports whether one of its requests was granted and if so, which target it can address in the following transfer stage.

The scheduler uses a proprietary algorithm called *Least Choice First Arbiter* [6]. This algorithm is not the focus of this paper and, therefore, described only briefly. The arbiter allocates connections between initiators and targets based on the number of requests. More specifically, when a target is scheduled, the initiator with the smallest number of requests is given highest priority since it has the fewest choices. To avoid starvation, a round-robin scheme is added that guarantees that a request is scheduled in bound time.

The timing of the channels is depicted in Figure 5. In this example two pairs of initiators  $bini$  and  $qini$  generate packets for the bulk channel and the quick channel, respectively. The stages of the bulk pipeline can be easily identified. Looking at the transfer of bulk request packet  $breq^n_{0,1}$ , slot  $s_{n-1}$  corresponds to the arbitration stage, slot  $s_n$  to the transfer stage, and slot  $s_{n+1}$  to the acknowledgment stage. The sequence of packets is as follows. In slot  $s_{n-1}$ , configuration packets  $cfg^n_{0,s}$  and  $cfg^n_{1,s}$  are sent from  $bini_0$  and  $bini_1$ , respectively, to the switch; when arbitration is finished, grant packets  $gnt^n_{s,0}$  and  $gnt^n_{s,1}$  return the results:  $bini_0$  is granted the request for  $btgt_1$  and  $bini_1$  is granted the request for  $btgt_0$ . In slot  $s_n$ , the corresponding request packets  $breq^n_{0,1}$  and  $breq^n_{1,0}$  are transferred. Finally, in slot  $s_{n+1}$ , the acknowledgment packets are transferred, that is,  $btgt_0$  sends  $back^n_{0,1}$  to  $bini_1$  and  $btgt_1$  sends  $back^n_{1,0}$  to  $bini_0$ .

A transmission error is detected if a negative acknowledgment is received or the acknowledgment is missing. A negative acknowledgment is returned if no receive buffer is available. (If the target detects a CRC

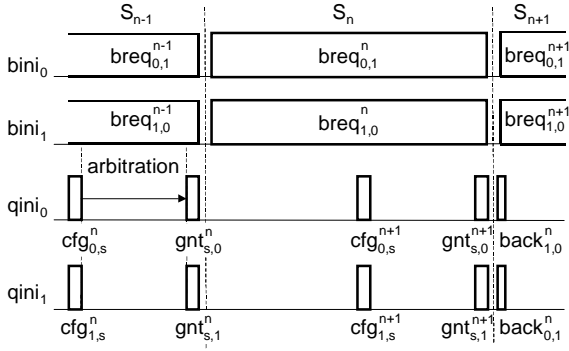


Figure 5: Channel Timing.

error, it does not return an acknowledgment to avoid the possibility of returning the acknowledgment to the wrong initiator.) The loss of a packet is detected if the initiator does not receive an acknowledgment a fixed amount of time after the request was sent.

When the quick switch forwards packets, it gives the acknowledgment packets *back* and *qack* as well as the scheduling packets *cfg* and *gnt* priority over *req* by dropping *req* in the case of collisions. Since the initiators know the times when grant packets are sent, they can avoid collisions by not sending any packets during these times. Collisions between the high-priority packets *back*, *qack*, *cfg*, and *gnt* are not possible for the following reasons. The configuration and grant packets are exchanged between the nodes and the switch, and do not actually pass through the switching fabric. Though the bulk acknowledgment packets do pass through the switching fabric of the quick switch, they cannot collide since the paths used correspond to the inverted schedule that was originally calculated for forwarding the bulk request packets in the bulk switch. The schedule is said to be inverted as the request and acknowledgment packets use the switch paths in opposite directions. For example, in Figure 5  $breq_{0,1}^n$  is forwarded from input port 0 to output port 1, and the corresponding acknowledgment packet  $back_{1,0}^n$  is forwarded from input port 1 to output port 0. Thus, bulk acknowledgment packets are never dropped and bulk request packets cannot be lost as a result of collisions on the quick channel.

For the Clint prototype, the transmission of the acknowledgment and scheduling packets consumes 5% of the quick channel bandwidth leaving ample bandwidth for regular data packets.

Since the grant packets are sent simultaneously and at a fixed time relative to the bulk slot boundaries, they are further used to synchronize the nodes. Slot boundaries of different nodes are, of course, not perfectly aligned and differences in transmission delays, for example, caused by differences in link lengths, have to be considered. While in our prototype implementation a gap is

inserted at the beginning of a slot to compensate for the misalignment, it would be possible to improve the alignment by actually measuring the round-trip times of the links.

The grant packets are also used to assign a unique identifier to each node. The node identifier corresponds to the number of the output port from where the packet was sent.

### 3.4 Quick Channel

A best-effort approach is applied to the quick channel in that packets are sent as soon as they become available. As a result, collisions occur in the quick switch in which case packets are dropped and retransmitted. Dropping packets is not as drastic a measure as might appear. Collisions happen infrequently since the quick channel is only lightly loaded due to the provision of excess bandwidth. While typical applications are expected to generate only light load on the quick channel, a throttling mechanism could be added to the nodes to limit usage of quick channel bandwidth. Retransmissions do not add a burden to the hosts since the NIC hardware already contains the necessary control logic and buffers to handle transmission errors.

The quick switch uses a minimal scheduler that applies a first-come, first-considered policy. When a packet arrives at an input port, a request is sent to the output port specified by the routing information contained in the packet header. When quick packets collide in the switch, the packet that arrives first will win and be forwarded and packets that arrive later will lose and be dropped. If colliding packets arrive simultaneously, a round-robin scheme is used to pick the winner and the losers. This way starvation is avoided. Further, this scheduler is able to make quick routing decisions so that forwarding latencies are kept low.

### 3.5 Implementation

Figure 6 shows the prototype board of the Clint switch. The bulk channel and quick channel can be easily identified. The bulk switch is implemented with an AMCC S2018 17 x 17 crosspoint switch and the quick switch is realized with a Xilinx XCV600 FPGA and four AMCC S2064 quad serial transceivers.

The main components of the NIC are a Xilinx XCV2000 FPGA, three Micron MT54V512H18 9 Mbit QDR SRAMs implementing the send and receive buffers, a TI TLK2500 serial transceiver interfacing the bulk channel, and an AMCC S2064 serial transceiver interfacing the quick channel.

In the following, we briefly want to discuss the major challenges we encountered during the development of the Clint prototype systems.

#### *Number of Clocks*

The FPGA of the NIC uses a total of five clocks:

- PCI clock (66/132/264 MHz): This clock is provided by the PCI bus and clocks the PCI interface logic. It also serves as the main clock that drives the majority of the NIC logic. 2x and 4x versions of this clock are used for interfacing the QDR SRAMs.
- Quick channel send clock (66 MHz): This is the send clock for the quick channel transceiver.
- Quick channel receive clock (66 MHz): This clock is recovered from the incoming data stream by the quick channel transceiver.
- Bulk channel send clock (125 MHz): This clock supplies the send clock for the bulk channel transceiver.
- Bulk channel receive clock (125 MHz): This clock is recovered from the incoming data stream by the bulk channel transceiver.

The large number of clocks posed several problems. Since the FPGA only provides four dedicated clock networks, we had to use regular wiring for some of the clocks. As a result, skew control became a critical issue for those clock networks implemented with regular wiring. Further, crossing clock domains required synchronizers in the form of FIFOs. We designed these FIFOs carefully to reduce the possibility of metastable

states and to avoid over- and underflow.

#### *PLL Synchronization*

Since we use an analog switch for the bulk channel, the transceiver of the receiving NIC has to be resynchronized with the clock of the sending NIC if the schedule changes; possibly, this happens every slot. Unfortunately, transceiver PLLs are typically not designed to quickly lock onto the clock embedded into the received data stream - locking, typically, happens only when the system is powered up. While the TI TLK2500 transceiver has the shortest locking time of the available transceivers, the overhead of the receiver PLL locking onto the received data still constitutes 5% of a slot.

#### *Links*

We decided to use coax cables for both the quick and the bulk links as they provide significantly lower cost than optical links. Though the high transmission rate of 2.5 Gbit/s limits the span of coax cables to roughly 5 m, this link length seems sufficient for a rack-mounted cluster with a small to medium size.

The challenges when designing the 2.5 Gbit/s bulk link is the attenuation as well as the mismatches in delays when transmitting over differential wires. Both factors impact the size of the signal eye seen by the receiving transceiver. We tested several different cables and finally determined that the SkewClear 150 Ohm Twinax cable from SpectraStrip performed the best. We chose a low AWG of 22 to minimize attenuation. With a loss of 0.58 db/m, a 5 m cable contributes a loss of 2.9 db.

When looking at the overall attenuation and delay

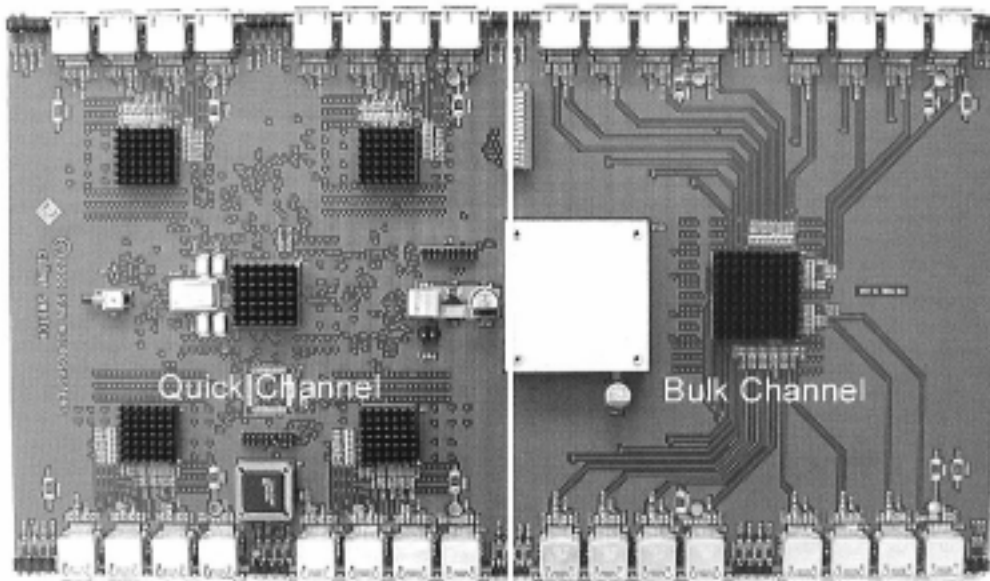


Figure 6: The Prototype Board of the Clint Switch

mismatch of a link, the PCB traces and cable connectors also need to be considered. We chose Berg HSSDC connectors specified for operating frequencies of up to 2.5 Gbit/s. We estimate that each connector adds a loss of 1 db. Finally, we manually laid out the differential PCB traces to minimize attenuation and control their impedance by choosing the trace geometry accordingly. Given a maximal trace length of 6 in we calculated the attenuation of the trace to be 3 db = 6 in x 0.5 db/in. We avoided vias as much as possible since they pose a significant impedance mismatch. Measurements revealed that the characteristic impedance of the differential traces is 110 Ohm rather than 150 Ohm. We found that the impedance calculations based on the geometry of the traces mistakenly did not consider the fact that the thickness of the copper traces on the outer layers increases during the metal plating process used to create vias.

#### 4. Simulation Results

In this section we analyze the Clint architecture with the help of a simulation model based on a register-transfer level description.

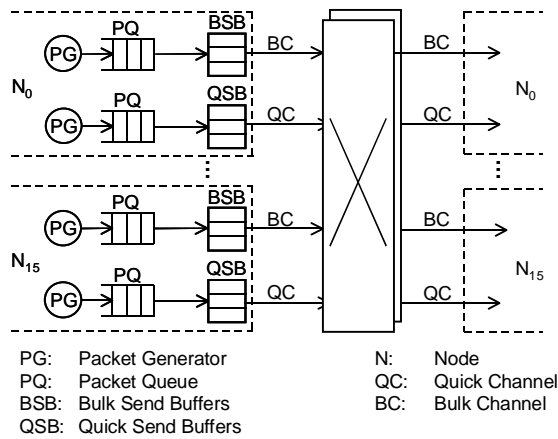


Figure 7: Simulation Model.

#### 4.1 Simulation Methodology

Figure 7 shows a block diagram of the simulation model. A cluster with 16 hosts is considered. For each channel, an initiator contains a packet generator that deposits packets into a packet queue that looks like a FIFO queue<sup>3</sup> with unlimited length. Packets are transferred from the packet queue to the send buffers when space permits. The number of send buffers is a parameter of the simulation run.

<sup>3</sup> Future work will have to address the benefits of out-of-order prefetching as described in Section 3.1.

The timing and behavior of the two channels is modeled as realistically as possible. Timing resolution is a cycle of 4 ns which corresponds to the byte-time on the bulk channel. A simulation run takes  $10^7$  cycles.

Latency refers to the time a packet spends in the packet queue as well as in the buffer pool. More specifically, the latency time starts when the packet is generated and deposited into the packet queue and stops when transmission of the packet out of the send buffer begins. The latency numbers neither include wire delays nor serialization delays, which can be a dominant factor for small latency values.

Latency times are calculated as a function of load. The load refers to the amount of packets generated by the packet generator. It is given as the fraction of the available link bandwidth.

We are using a synthetic workload with the following characteristics. The destinations of the packets form a uniform distribution. The mean value of the injection rate maintained by the packet generator is a parameter of the simulation while the injection interval varies according to a uniform distribution. An option is provided to generate packets in bursts; again, the distribution of burst lengths is uniform.

We are currently porting Sun HPC ClusterTools 4.0<sup>TM</sup> to be able to benchmark MPI applications running on Clint. Still, a simulation of a synthetic workload has its own value as it more easily allows for evaluating individual architectural features.

#### 4.2 Latency Simulations

Figure 8 illustrates the results of the latency simulations for the bulk channel. Four different architectures are considered:

- B1. The bulk channel is not scheduled. The bulk switch contains no packet buffers. If packets collide, one packet is forwarded and the other ones are dropped and retransmitted.
- B2. The bulk channel is not scheduled. The bulk switch contains no packet buffers. Collisions are avoided by applying back pressure.
- B3. The bulk channel is not scheduled. Collisions are resolved by storing packets in output buffers of the switch. If an output buffer overflows, packets are dropped and later retransmitted.
- B4. The bulk channel is globally scheduled. This organization corresponds to the Clint architecture described in the previous section.

For each architecture, the number of send buffers contained in the hosts is 16. The output-buffered switch has 16 buffers per output port. A buffer is large enough to hold a complete packet.

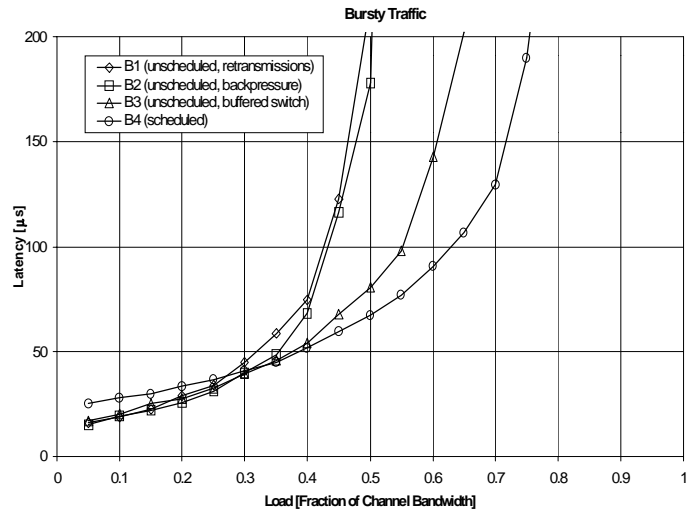
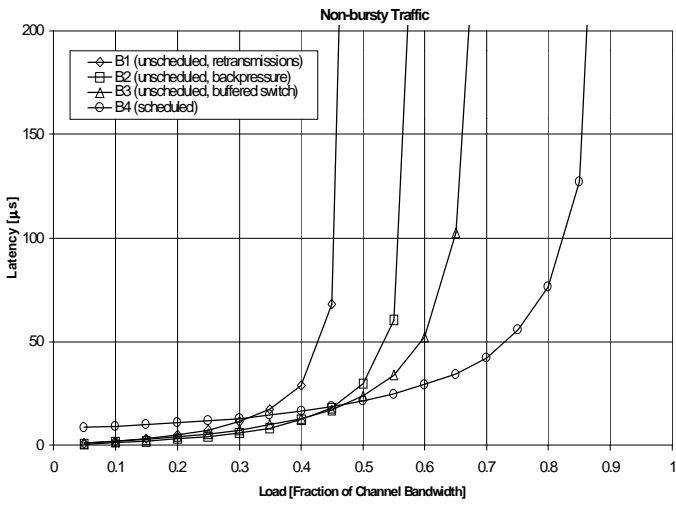


Figure 8: Latency Simulations for the Bulk Channel.

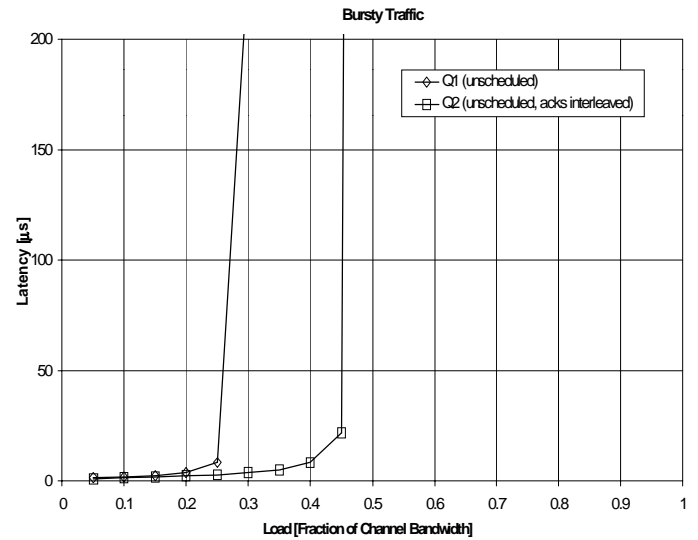
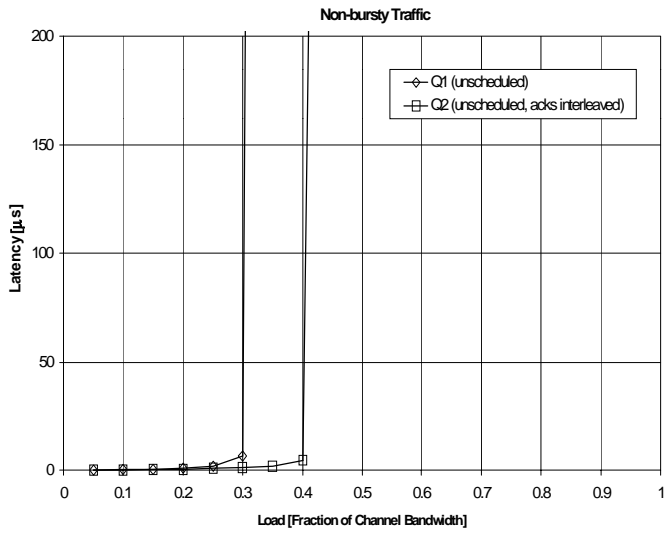


Figure 9: Latency Simulations for the Quick Channel.

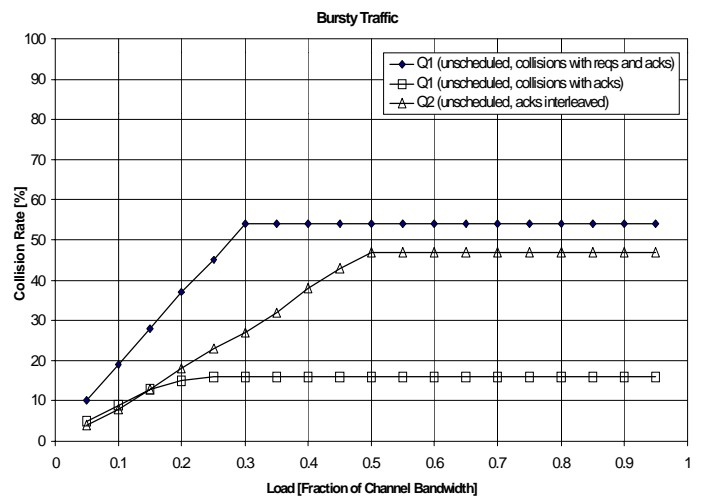
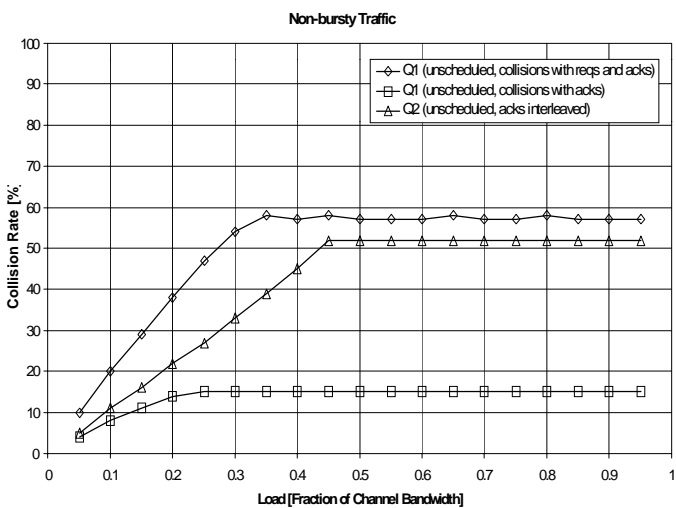


Figure 10: Collision Rates for the Quick Channel.

On the bulk channel, the load is made up entirely of bulk request packets. The simulation assumes error-free transmission on the bulk channel. Bursty and non-bursty traffic patterns are examined. Bursts have a maximal length of five packets.

Examining the graphs of Figure 8 the following observations can be made. As the load increases, the latency increases less for the scheduled bulk channel than for the unscheduled versions of the bulk channel. In fact, the graphs show that the unscheduled architectures saturate at much less than the maximal channel capacity. In these cases channel utilization is limited by HOL blocking [9]. In contrast, the scheduled Clint channel can be operated well above this point. Comparing architectures B1 and B2 the use of back pressure reduces latency since it takes less time to recover from a conflict. Adding buffers to the switch also improves channel utilization, however, it cannot remove HOL blocking which is a problem inherent to an unscheduled switch.

Architectures B1-B3 represent conventional networks that rely on local switching decisions. As the graphs clearly show, the use of a global schedule by the Clint architecture B4 yields a significant improvement in channel utilization.

There is, however, a price to pay for scheduling a channel and the resulting higher channel utilization. Imposing globally synchronized time slots to transmit packets adds, on average, half a slot to the packet latency. The latency is further increased by the time needed to calculate the schedule. For these reasons, the latencies are higher for the scheduled bulk channel B4 than for the unscheduled bulk channels B1-B3 at low load. However, it should be kept in mind that the bulk channel is not optimized for low latency and that low-latency packets are to be sent over the quick channel.

We have developed a method to speculatively schedule the bulk schedule to reduce the synchronization penalty under light load. We will describe this method in a future publication.

Figure 9 illustrates the simulation results for the quick channel. Two architectures are considered:

- Q1. This is the unscheduled quick channel of the Clint architecture. Packets may be dropped and retransmitted as a result of collisions.
- Q2. This variation of the Clint quick channel interleaves quick request and acknowledgment packets in the switch and the NIC. That is, rather than dropping request packets that collide with acknowledgment packets, the switch intersperses acknowledgment packets into the request packets. And the NIC interleaves the transmission of acknowledgment and

request packets rather than delaying the transmission of either of them.

In both scenarios the hosts contain four send buffers.

We have simulated the latencies for the quick request packets again as a function of load. In the case of the quick channel the load is made up of different types of packets. They are modeled as follows. The effect of the bulk acknowledgment packet, configuration packet, and grant packet on the simulated latencies are modeled as a reduction of the available bandwidth. The transmission of both the quick request and acknowledgment packets is modeled in detail including collisions. Figure 10 only shows the load created by request packets – the shown load does not include the traffic caused by quick acknowledgment packets or packet retransmissions.

Analyzing the graphs, the following observations can be made. The latencies are an order of magnitude smaller for the quick channel than for the bulk channel. The quick channel does, however, saturate under less load than the bulk channel. The reason, of course, is the lack of scheduling that would avoid HOL blocking and collisions. The quick channel does, however, provide enough bandwidth that we do not expect it to be operated anywhere close to the saturation point. In terms of packet throughput, the quick channel provides 15 times the capacity of the bulk channel, that is, for every request forwarded on the bulk channel, 15 requests are forwarded on the quick channel. Given that link saturation is observed around 35% link capacity a ratio of up to 5 quick request packets per bulk request packet can be tolerated.

As the graphs show channel utilization can be significantly improved by interleaving request and acknowledgment packets. Interleaving is applied in the switch as well as in the NIC. In the switch, packet interleaving reduces collisions and, with it, latency as request packets are no longer dropped when they collide with acknowledgment packets. In the NIC, packet interleaving reduces latencies as the transmission of acknowledgment packets is no longer delayed when a request packet is being sent.

We have extracted data points of graphs B4 and Q2 and assembled them in Table 2 to make a quantitative comparison easier. For bursty traffic, the latency times for the quick channel are 19 and 11 times smaller for loads of 0.1 and 0.3, respectively, than for the bulk channel.

Figure 10 shows the collision rates for architectures Q1 and Q2. For Q1 we determined the rate of collisions with acknowledgment packets in addition to the rate of collisions with both request and acknowledgment packets. For low load there are roughly as many collisions with request packets as there are with acknowledgment

		Latency [ $\mu$ s]		Latency Ratio
		Bulk Channel	Quick Channel	
Load=0.1	Non-bursty	7.4	0.1	51
	Bursty	21.0	1.1	19
Load=0.3	Non-bursty	10.5	1.0	11
	Bursty	32.9	2.8	11
Load=0.5	Non-bursty	16.8	saturated	
	Bursty	53.5		
Load=0.7	Non-bursty	34.2		
	Bursty	103.7		
Load=0.9	Non-bursty	464.4		
	Bursty	1636.7		

Table 2: Selected Latency Simulations.

packets. As the load and, with it, the number of retransmission increases, the ratio of request packets and acknowledgment packets changes such that there are many more request packets. As a result, collisions with request packets are more likely than collisions with acknowledgment packets.

## 5. Related Work

Similarly segregated network architectures can also be found in parallel machines such as the Cray T3D [10] and the CM-5 [13]. These machines contain a low-latency network mainly used for synchronization operations in addition to a general-purpose network. Compared with Clint, the physical span of these networks is, however, much more limited.

The use of multiple networks with different characteristics is examined in [11],[12]. In these papers, a technique called performance-based path selection is tested on a cluster of SGI multiprocessors interconnected with Ethernet, FibreChannel, and HIPPI networks. This technique selects the network that offers the lowest latency for the size of the message to be transmitted. Speedups of up to 2.11 are measured for the selected benchmark programs.

A hybrid router architecture consisting of one switch using wormhole routing and several switches using circuit switching is described in [4]. The former switch implements several virtual channels of which a subset is used to set up and tear down the physical circuits implemented by the latter switches. The nodes pre-establish circuits by sending probes to the routers. Unlike the central bulk scheduler of Clint, the routers only have partial knowledge of the packets waiting in the nodes to be sent and, as a result, throughput is reduced accordingly.

The use of separate mechanisms to transport short and long messages has been proposed by the Illinois Fast

Messages (FM) protocol [18] and the Scheduled Transfer Protocol (ST) [19] for HIPPI-6400. The FM protocol provides two send routines similar to sending a bulk packet and a quick packet, respectively. Unlike Clint, both routines use the same channel. ST uses two separate channels: A high-bandwidth data channel and a low-latency control channel. The control channel is used by upper-layer protocols and is not available for the exchange of low-latency user-level messages.

It is important to recognize the differences between the segregated architecture of Clint and other architectures with multiple channels such as HIPPI-6400 [19]. While Clint implements its two channels with physically separate networks, that is, with separate links and switches, other approaches multiplex multiple channels onto a single shared network. Some degree of decoupling can be achieved by using separate buffer queues [3]. This technique is quite efficiently used to prevent HOL blocking in input-buffered switches [9]. Still, the channels are competing for shared resources such as links or switching fabrics. Also, design objectives set forth for each channel have to be compromised if there is one physical channel. Another important difference is the usage model for the channels. Both channels of Clint are intended for general-purpose usage. In particular, both channels are available to user programs. This is not true for networks such as HIPPI-6400 that provides a control channel in addition to a regular data channel and that restricts the usage of the control channel to protocol processing.

## 6. Conclusions

We have described a segregated network architecture that provides two physically separate channels with different characteristics: A high-bandwidth channel for the scheduled transfer of large packets at high bandwidth and a low-latency channel for best-effort delivery of small packets with low forwarding latency. Only by separating these two concerns, a network can be obtained that achieves high throughput and, at the same time, low latency. The simulation results show that the bulk channel can be utilized nearly up to its full capacity and the quick channel provides latencies that are an order of magnitude smaller than for the bulk channel.

Future work will have to explore criteria for selecting the channels. If the choice is not left to the application, the characteristics of the message such as size or type might be used as a selection criteria. For example, short messages containing synchronization operations and resource management operations could benefit from using the quick channel.

We have shown the advantages of a globally scheduled bulk channel that considers the packets as soon as

they become available in the nodes: It achieves higher network utilization as well as lower latencies under load than other unscheduled networks or networks with scheduling local to the switch. This was achieved by imposing a rigid timing regime that relies on fixed forwarding delays and coordinated packet transmission. This allows transmission paths to be allocated at the time a packet leaves a host so that packets passing through the network do not inflict any collisions. By avoiding such conflicts, head-of-line blocking is avoided.

As the bulk channel requires only a simple flow-through switch that does not have to process packets, it can be easily scaled to higher speeds. Furthermore, an implementation can be envisioned that uses a purely optical interconnect for the bulk channel and an electrical implementation for the quick channel.

### Acknowledgments

Neil Wilhelm helped with the initial design of Clint. Alan Mainwaring specified the original version of the NIC. Nicolas Fugier, Marc Herbert, Éric Lemoine, and Bernard Tourancheau are interfacing Clint to Sun HPC ClusterTools™. Scott Powers and José Cruz conducted elaborate measurements of the physical layer.

### References

- [1] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W. Su: *Myrinet: A Gigabit-per-second Local Area Network*. IEEE Micro, 15(1):29--36, February 1995.
- [2] B. Chun, A. Mainwaring, D. Culler: *Virtual Network Transport Protocols for Myrinet*. IEEE Micro, vol. 18, no. 1, January/February 1998, pp. 53-63.
- [3] W. Dally: *Virtual-Channel Flow Control*. Proc. of the 17th Int. Symposium on Computer Architecture, ACM SIGARCH, vol. 18, no. 2, May 1990, pp. 60-68.
- [4] J. Duato, P. López, F. Silla: *A High Performance Router Architecture for Interconnection Networks*. Proc. Int. Conf. On Parallel Processing, 1996.
- [5] D. Gustavson: *The Scalable Coherent Interface and Related Standards Projects*. IEEE Micro, vol. 12, no. 1, Feb. 1992, pp. 10-22.
- [6] N. Gura, H. Eberle: *The Least Choice First Scheduling Method for High-Speed Network Switches*. IPDPS 16<sup>th</sup> Int. Parallel and Distributed Processing Symposium, Fort Lauderdale, April 15-19, 2002.
- [7] J. Hoffman: *HIPPI-6400 Technology Dissemination*. Proc. of SPIE, vol. 2917, 1996, pp. 442-430.
- [8] R. Horst, D. Garcia: *Servnet SAN I/O Architecture*. Hot Interconnects V Symposium, Stanford, Aug. 21-23, 1997.
- [9] M. Karol, M. Hluchyi, S. Morgan: *Input versus Output Queuing on a Space-Division Packet Switch*. IEEE Transactions on Communications, C-35(12):1347-1356, December 1987.
- [10] R. Kessler, J. Schwarzmeier: *Cray T3D: A New Dimension for Cray Research*. Proc. 38<sup>th</sup> IEEE Int. Computer Conf., 1993, pp. 176-182.
- [11] J. Kim, D. Lilja: *Utilizing Heterogeneous Networks in Distributed Parallel Computing Systems*. Proc. of the 6<sup>th</sup> Int. Symposium on High Performance Computing, 1997.
- [12] J. Kim, D. Lilja: *Performance-Based Path Determination for Inter-processor Communication in Distributed Computing Systems*. IEEE Trans. on Parallel and Distributed Systems, vol. 10, no.3, March 1999, pp.316-327.

- [13] C. Leiserson et al.: *The Network Architecture of the Connection Machine CM-5*. Proc. 1992 Symp. Parallel Algorithms and Architectures, 1992, pp. 272-285.
- [14] A. Mainwaring: *Active Message Application Programming Interface and Communication Subsystem Organization*. University of California at Berkeley, Computer Science Department, Technical Report UCB CSD-96-918, October 1996.
- [15] A. Mainwaring and D. Culler: *Design Challenges of Virtual Networks: Fast, General-Purpose Communication*. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP), Atlanta, Georgia, May 4-6, 1999.
- [16] N. McKeown, C. Calamvokis, S. Chuang: *A 2.5 Tb/s LCS Switch Core*. Hot Chips 13, August 19-21 2001, Stanford, California.
- [17] Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard*. Technical Report 94-230, Computer Science Department, University of Tennessee, May 1994.
- [18] S. Pakin, V. Karamcheti, A. Chien: *Fast Messages (FM): Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processor*, IEEE Concurrency, vol. 5, no. 2, 1997, pp. 60-73.
- [19] National Committee for Information Technology Standardization: *Scheduled Transfer Protocol (ST)*. Task Group btgt1 1.1, rev. 3.6, January 31, 2000, www.hippi.org.
- [20] V. Sunderam: *PVM: A Framework for Parallel Distributed Computing*. Concurrency: Practice and Experience, vol. 2(4), December 1990, pp. 315-339.

Sun, Sun Microsystems, the Sun logo, Sun HPC ClusterTools are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.