StarT-X
A One-Man-Year Exercise in Network Interface Engineering

Computation Structures Group Memo 406
July 21, 1998

James C. Hoe
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
jhoe@lcs.mit.edu

In Proceedings of Hot Interconnects 6

This paper describes research done at the MIT Laboratory for Computer Science. Funding for this work is provided in part by the Defense Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research contract N00014-92-J-1310 and Ft. Huachuca contract DABT63-95-C-0150.

StarT-X

A One-Man-Year Exercise in Network Interface Engineering

James C. Hoe*
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
jhoe@lcs.mit.edu

Abstract

This paper presents the StarT-X PCI card, a network interface unit for the Arctic Switch Fabric[4]. StarT-X provides a user-level hardware interface for message passing on a cluster of PCI-equipped host platforms. StarT-X supports three message-passing mechanisms that are tuned for different granularities of communication. On a SUN E5000 with StarT-X, a processor can send and receive a 64-byte message in less than 0.4 and 3.5 usec respectively and incur less than 5.6 usec user-to-user latency. StarT-X's remote memory-to-memory DMA mechanism can transfer large data blocks at over 60 MByte/sec on SUN E5000's. StarT-X's hardware was developed in just over a year by a one-man team. StarT-X and Arctic are currently installed on MIT/LCS's Xolas Cluster of SUN E5000's and support MPI[15] and Cilk[18] programming interfaces. The performance of these high-level programming interfaces and a numerical application, MITMatlab[12], are reported.

1 Introduction

The StarT-X PCI card is an NIU (Network Interface Unit) for the Arctic Switch Fabric[4], an experimental system area network. StarT-X provides a user-level hardware interface for message passing on a cluster of PCI-equipped host platforms. StarT-X is not designed for NIU research, but rather, is engineered to be used by a local community of computer and computational scientists on a cluster of SUN E5000 8-UltraSPARC SMP's (Symmetric Multiprocessors). Despite being a one-man effort, this project took just over a year from inception to running MPI[15] and Cilk[18] parallel applications.

StarT-X is one of the latest developments in a series of StarT[1, 5, 16] parallel processing cluster projects. As a successor to StarT-Jr[10], StarT-X shares many of StarT-Jr's message-passing mechanisms. However, StarT-X achieves significantly better performance than StarT-Jr in both bandwidth and latency by handling critical operations in hardware instead of embedded processing.

Salient features of StarT-X are:

- Three message-passing modes
  1. Memory-mapped message queues with programmed I/O interface
  2. Cacheable virtual message queues implemented in the host memory
  3. Remote memory-to-memory DMA transfer with automatic packetization
- Two message priorities
- Option of FIFO or non-FIFO[1] ordered message delivery

This paper presents the StarT-X NIU and its performance on a cluster of SUN E5000 SMP's. Section 2 first discusses the forces that motivated and constrained the StarT-X project. To put StarT-X in context, Section 3 describes two commercial interconnect technologies that were also considered for the SUN cluster. Section 4 presents the StarT-X datapath. Section 5 explains the three StarT-X message-passing mechanisms. Section 6 reports StarT-X related software developments and their performance. Section 7 concludes with a few remarks regarding the development of StarT-X.

*Funding for this work is provided in part by the Defense Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research contract N00014-92-J-1310 and Ft. Huachuca contract DABT63-95-C-0150.

[1]To utilize randomized up-route in the Arctic Switch Fabric's fat-tree topology.

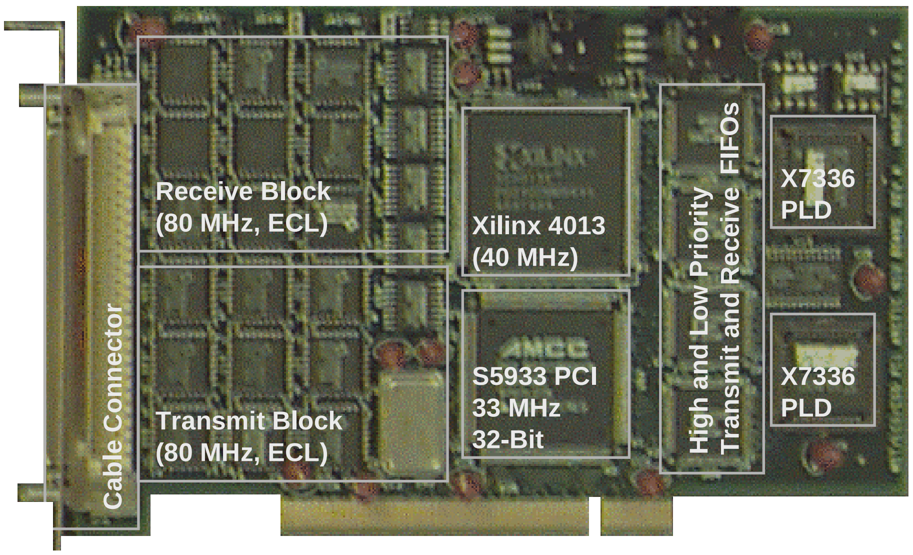

Figure 1: StarT-X PCI Short Card

2 Background and Constraints

The StarT-X project rose out of the demand for better intra-cluster communication beyond the stock 100-Mbit Ethernet on a cluster of SUN E5000 SMP's. The Arctic Switch Fabric was put forth as a high performance alternative. The Arctic Switch Fabric is a packet-switched binary fat-tree network with a bi-directional bandwidth of 2.4 Gbit/sec on each link. However at the time, Arctic did not have a suitable adapter to interface to a SUN E5000. This prompted us to investigate the practicality of developing StarT-X to fill this gap.

Our conversations with the cluster's users revealed that an NIU like StarT-X would be useful only if it could be developed quickly. A prolonged development time would make StarT-X unattractive because several commercial alternatives, like Myrinet, SCI, ATM, etc. were readily available. Furthermore, the SUN E5000 cluster itself would become obsolete in only a few years. Given these considerations, the StarT-X project was launched with a strict 12-month time constraint.

Since the issues in interfacing Arctic to a host had been studied in two other Arctic NIU designs[1, 10], StarT-X could leverage many design overlaps instead of working from scratch. Thus, only one full-time staff was assigned to the project despite the time pressure. A deliberate effort was made to avoid turning StarT-X into a research project. The foremost goal is to quickly engineer a flexible, high-performance NIU, under a level of effort and risk that is justifiable by the advantages Arctic has to offer. To further ensure many returns on this investment, we included platform-independence, i.e. PCI[17], as a requirement so StarT-X could also target other clusters.

3 Clustering Technologies

Besides standard local area networks, various specialized interconnection technologies are available to construct parallel processing clusters from all major platforms of stand-alone PC's or workstations. This section describes two commercial high-performance clustering hardware that is compatible with SUN Enterprise servers.

SUN offers a clustering package to support key applications on a cluster of servers interconnected by Dolphin SCI (Scalable Coherent Interface)[7]. Current Dolphin SCI adapters support non-cacheable remote read and write operations and are available for either SBus (32-bit) or PCI (32-bit, 33 MHz). Four-port switch units, with 6.4 Gbit/sec per port, can be composed to connect up to 16 endpoints. To explore the full performance of the SCI hardware, Ibel et al.[13] implemented Active Message and Split-C for lightweight user-level communication. On two SUN Ultra-1 workstations connected back-to-back with SBus adapters, their programming interface achieved a round-trip latency as low as 13.3 usec and bandwidth up to 25 MByte/sec.

Myricom supplies another high-performance clustering package for SUN UltraSPARC platforms. The Myrinet[3] package comes complete with network routers, end-point adapters for SBus (32-bit) or PCI (32-bit, 33 MHz), standard IP layers, and a cus-

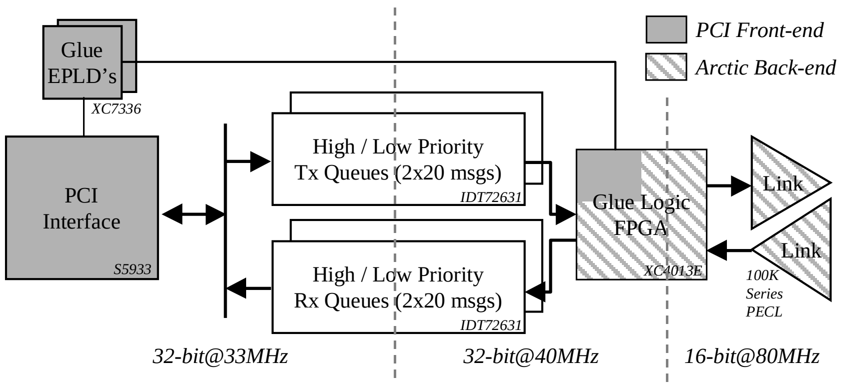

Figure 2: StarT-X Datapath

tom low-level interface layer. An arbitrary network topology can be constructed from a selection of 4, 8 and 12-port switches. Myrinet adapters are controlled by Myricom's custom LANai embedded processors. Several research projects make use of the adapter's programmability to experiment with different interface schemes for fast user-level message passing. On a cluster of SUN Ultra-1 workstations with SBus adapters, Culler et al.[14] have reported a round-trip latency of 17.7 usec and peak bandwidth of 33 MByte/sec using Active Message and custom LANai firmware.[2]

4 StarT-X Datapath

The StarT-X NIU (Figure 1) conforms to the PCI[17] short card (4.2" by 6.6") standard. To complete the development within a limited time frame, we designed a datapath based on a simple integration of off-the-shelf parts plus a small number of programmable logic devices. As depicted in Figure 2, the datapath can be organized into four functional categories: PCI interface (front-end), message queues, Arctic link interface (back-end), and glue logic.

PCI Interface:
At the front-end, StarT-X incorporates the S5933 PCI Interface chip from AMCC[2]. The interface shelters the internal StarT-X datapath from the electrical and logical issues of interfacing to a 32-bit 33 MHz PCI bus. The S5933's PCI slave interface and two DMA engines serve as the basis for StarT-X's message-passing mechanisms. Although SUN E5000 supports 64-bit 66 MHz PCI, StarT-X could not target for it within the development time frame. StarT-X would have to develop a custom ASIC to support 66 MHz or a custom PCI interface design to support 64-bit PCI with an FPGA at 33 MHz.

[2]Working with the PCI version of Myrinet adapters on Intel or Digital platforms, other projects have seen higher performance on Myrinet. (See http://www.myri.com)

Message Queues: StarT-X employs four IDT723631 synchronous FIFO's to implement two transmit and two receive message queues. Each queue holds a maximum of twenty messages (up to 96 bytes per message). These on-board buffers smooth the effects of transient throughput mismatch between the PCI and Arctic link interfaces. The queues also provide synchronization for data crossing between the front-end's 33 MHz PCI clock domain and the back-end's 80 MHz/40 MHz network clock domain.

Arctic Link Interface:
An Arctic link achieves 160 MByte/sec using 16 differential PECL signals in parallel clocked at 80 MHz. An Arctic cable bundles two links running in opposite directions. A 5V-CMOS/PECL conversion circuitry sits between StarT-X's front-end and the link cable connector. The circuitry is implemented using 24 100K Series PECL chips that occupy half of the board space. The link interface is an example where StarT-X derived overlapped designs from other Arctic NIU's to save on development time.

Custom Glue Logic:
The glue logic is implemented using a Xilinx 4013E-2 FPGA and two Xilinx 7336-5 EPLD's. The fast 5-nsec EPLD is necessary to operate S5933 at its maximum throughput. The glue logic design is captured in RTL-level Verilog for synthesis by the Synopsys Hardware Compiler and Xilinx's XACT tools. The glue logic is functionally sub-divided into a front-end and a back-end, but the two partitions are packed

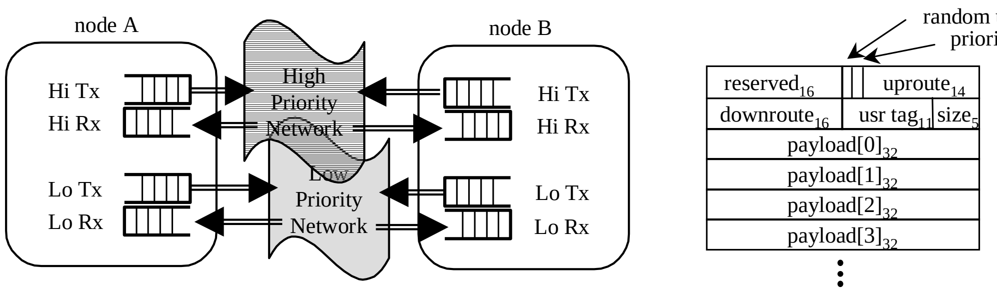

Figure 3: (a) PIO Mode Abstraction and (b) StarT-X Message Format

in a single FPGA to conserve board space. The front-end logic, synchronous to the 33 MHz PCI clock, mediates the interaction between the S5933 PCI interface and the message queues. The front-end logic also creates the different personalities of the three StarT-X message-passing mechanisms. The back-end logic sits between the message queues and the link interface. It checks or generates a 16-bit CRC for each message entering or leaving on the Arctic link. The back-end logic also provides multiplexing and demultiplexing between the 16-bit 80 MHz Arctic link interface and the 32-bit 40 MHz message queue interface.

5 StarT-X Mechanisms

In designing a platform-independent NIU, it is difficult to optimize all aspects of communication performance in a single mechanism. Instead, StarT-X provides three separate user-level message-passing mechanisms, each optimized for a different range of operations. These mechanisms are selected based on our experience with the StarT-Jr[10] cluster.

5.1 Programmed I/O (PIO) Mode

The PIO mode presents a simple FIFO-based network abstraction depicted in Figure 3(a). This memory-mapped mechanism is similar to CM-5's data network interface[19]. The PIO mode requires the user to directly manipulate the StarT-X hardware, and thus, has the highest processor overhead and the lowest bandwidth of the three mechanisms. However, this mechanism also has the simplest software interaction and achieves the lowest user-to-user latency. This mode is intended for fine-grain message passing where messages are short and frequent.

In this mode, the user communicates in the unit of a message. The StarT-X message format is shown in Figure 3(b). A message contains two 32-bit header words followed by a variable size payload between 2 and 22 32-bit words. A message is addressed by the receiver's ID in the down-route field. Specifying the up-route selects a deterministic path in a fat-tree network and ensures FIFO ordering of same-priority messages on the same path. Setting the random up-route bit in the header instructs StarT-X to randomly distribute network traffic on the up-route portion of the fat-tree for load balancing. StarT-X provides two pairs of transmit and receive queues to buffer high and low-priority messages separately. A priority bit in the header word indicates the message's priority and selects between one of the two virtual networks in the Arctic Switch Fabric. Congestion in the high-priority network can block messages in the low-priority network, but not vice versa.

To send a message, the user enqueues the message, word-by-word, directly into a transmit queue via PIO writes to a memory-mapped interface location. After the last word is enqueued, StarT-X automatically launches the message without requiring a separate command. The receiver can later retrieve this message with PIO reads from the receive queue's memory-mapped address. A memory-mapped status register indicates when new packets are waiting in the receive queues[3] and when a transmit queue is full.

A major handicap of PIO-based mechanisms is the cost of PIO accesses. On a SUN E5000, a 4-byte PIO read from a PCI register could stall the 166 MHz UltraSPARC-1 processor for 1.03 usec. This can be viewed as either 160+ cycles of processor overhead per word received or a maximum receive bandwidth of less than 4 MByte/sec. StarT-X applies two optimizations to speed up and reduce the number of PIO operations.

Whereas CM-5 maps its registers to unique addresses, each StarT-X interface register is redundantly mapped to all four-byte aligned locations of

[3]StarT-X can also raise an interrupt on the arrival of a new packet.

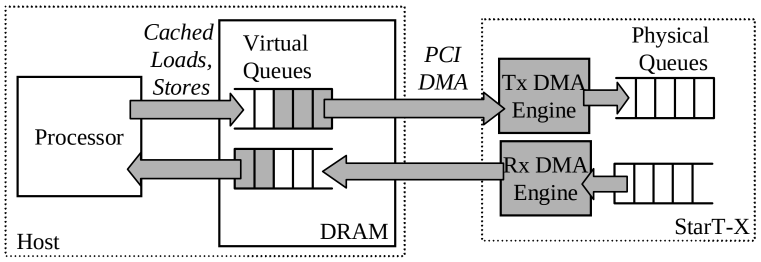

Figure 4: Cacheable Virtual Interface Message Queues

an entire virtual page. Thus, instead of repeatedly storing to the same address, the user stores to consecutive word addresses when enqueuing a message. This behavior triggers both the processor and the PCI bus bridge to merge multiple back-to-back stores into a single burst transaction to achieve better bandwidth. Although most hardware cannot merge PIO read transactions, the same optimization does allow the user to issue fewer PIO reads by issuing wider load instructions. For example, on SUN E5000, the user can receive a 64-byte message using a single 64-byte load (ldda). This optimization raises the peak PIO receive bandwidth from 4 to 32 MByte/sec (1.96 usec per 64-byte read).

Besides message reception, polling the status register accounts for the remainder of the costly PIO read operations. When sending a message, the user must check the status registers to avoid overflowing the transmit queue. The user must also read the status register to detect new messages in the receive queues. On CM-5, these conditions are only reported as true or false, and must be re-tested after each operation. To reduce the number of PIO reads to the status register, StarT-X reports the exact number of messages in the receive queues and the number of messages that the transmit queues guarantee to accept before overflowing. All four message counts are packed into a single status word so in the best case the cost of a status read can be amortized over 40 message sends and 32 receives.

5.2 Cacheable Virtual Interface (VI) Mode

Modern architectures are tuned for orders-of-magnitude higher performance for cached accesses to main memory than for PIO accesses. StarT-X's second mode of message passing takes advantage of this performance disparity. The VI mode virtually extends the high-priority transmit and receive queues into the memory system. Figure 4 illustrates this abstraction. The user interacts with StarT-X indirectly through memory, and hence, avoids costly PIO accesses.

The VI mode makes use of a pinned, contiguous physical memory region for DMA (direct memory access). The VI memory region is mapped into a cacheable virtual memory region of the user process. To send a high-priority message, instead of enqueuing directly to the hardware transmit queue, the user process only writes the message to the cacheable VI region. The user then invokes StarT-X's transmit DMA engine to enqueue the message into the physical transmit queue. The user does not need to explicitly flush the message from the processor cache because PCI DMA is cache-coherent. Multiple out-bound messages can be queued consecutively in memory and be transmitted with a single DMA invocation. This mode speeds communication by eliminating the PIO overhead associated with message handling. The user does incur a hefty overhead of a device driver call[4] to initiate a DMA request. Hence, transmitting in VI mode is only efficient in communication patterns where each DMA invocation can be amortized over a large batch of out-bound messages.

Receiving in the VI mode does not suffer from the same inefficiency and can be activated independently from the VI transmit mode. On systems that do not support extra-wide PIO-read instructions, a combination of PIO-mode transmit and VI-mode receive produces the best overall message-passing performance. When receiving, the user first reserves a large buffer (several megabytes) in the VI memory region. After the user programs the receive DMA engine with the base and the size of the receive buffer, subsequent incoming high-priority messages fill the buffer sequentially as they arrive without further user intervention. The user only needs to re-program the engine when the current receive buffer is exhausted. Extending the network buffers into physical mem-

[4]The device driver call guards against illegal physical memory accesses by user processes. This safety check is a requirement for StarT-X to co-exist in the cluster with other projects running production code. On our experimental Linux/PC cluster, DMA is activated by user processes directly.

ory greatly expands the effective network buffer size. This extra capacity can help prevent network congestion when a receiver is overwhelmed by a burst of incoming messages.

5.3 Remote DMA (RDMA) Transfer Mode

The RDMA mode is optimized for maximum bandwidth when transferring a large data block between two hosts. The hardware mechanism for the RDMA mode is the same as the VI mode with the addition of logic to packetize the RDMA data stream into messages and then to reconstruct the original data stream at the receive node. To initiate a RDMA transfer, the sender programs two StarT-X registers with the message header words and then invokes the transmit DMA engine to transfer data from main memory. StarT-X inserts the user-supplied message header words into the data stream at the appropriate intervals to feed a properly formatted message stream into the high-priority transmit queue. The receiving StarT-X reconstructs the original data stream by stripping the headers from the message stream and delivering the data stream to the receiver-specified memory locations.

A caveat to this mechanism is that the hardware formatted message stream does not contain sequencing or memory addressing information. The receiving hardware cannot reorder an out-of-order message stream or distinguish messages from multiple interleaved incoming streams. This simple mechanism depends on an ordered, uninterrupted high-priority transmission into the receiver's queue. Therefore, RDMA cannot use StarT-X's random up-route feature. Furthermore, RDMA senders must use an additional protocol to acquire exclusive right before transmitting to a receiver. The RDMA mode operation also cannot co-exist with other high-priority message-passing modes. RDMA operations do not interfere with low-priority message passing in the PIO mode.

6 Software Development and Performance

The first StarT-X cluster of four SUN E5000's came online in December, 1997. Since then, the project has focused on developing communication libraries to support end users' applications. This section presents the performance of several communication libraries at different levels of software abstraction. This section also presents our experience with a large MPI numerical application that has been ported to the StarT-X cluster.

6.1 JAM Communication Library

At the lowest level, the JAM kernel library provides a thin software veneer for C applications to access StarT-X's user-level hardware mechanisms. The hardware abstractions for the three StarT-X message-passing mechanisms are relayed directly to the user. The library does not add extra features through software. This low-level interface is intended to be the building block of higher-level communication libraries.

6.1.1 PIO Mode Performance

The characteristics of JAM PIO-mode primitives are studied using the LogP Signature microbenchmark devised by Culler et al.[6]. The experiment measures the time (T_{total}) required for a source processor (P_s) to send a sequence of m messages to a remote processor (P_r), pausing for d usec between each message. Each of the m messages generates a reply message from P_r. With finite network buffering, P_s may have to receive some reply messages before it can finish sending all m messages. A LogP signature is generated by plotting the average time (T_{total}/m) as a function of m for different d's. The send and receive overhead (O_s, O_r) and the gap (g)[5] can be extracted from the LogP signatures. Network latency ($L = \frac{T_{round-trip}}{2} - O_s - O_r$) requires a separate round-trip time measurement.

Figure 5 gives the LogP signatures of JAM PIO primitives in three different usages. Signatures (a) and (b) are for using a set of generalized send and receive primitives on 16-byte and 64-byte messages respectively. These primitives can transfer variable-length messages between any word-aligned memory buffer and the message queues. Signature (c) is for using a set of primitives specialized for 64-byte messages on SUN UltraSPARC systems. These primitives take advantage of `ldda` and `stda` instructions to transfer 64-byte messages between 64-byte-aligned memory buffers and the network queues. The experiments are conducted on two SUN E5000 8-processor SMP's. One 166 MHz UltraSPARC-1 processor in each SMP serves as P_r or P_s. In all cases, P_s and P_r exchange high-priority messages along a fixed network path over two router stages. Each data point

[5]The LogP Signature normally determines the gap which is related to the single worst bandwidth bottleneck on the path between a sender and a receiver. However, if $g < (O_r + O_s)$, the time for P_r to bounce each message appears as an artificial gap in the experiment. A different experiment has shown $g = O_r$ for StarT-X/Arctic.

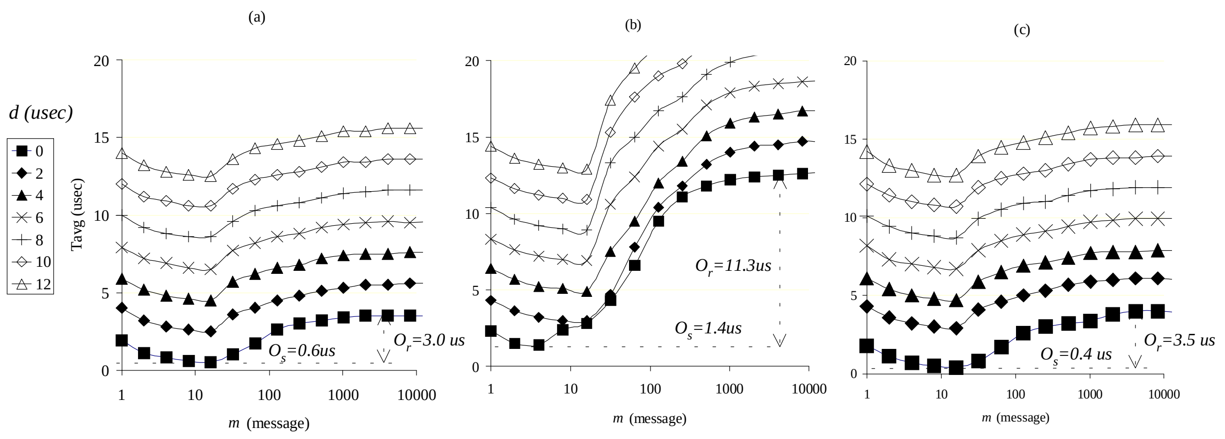

Figure 5: LogP Signatures of PIO Message Passing: (a) 16-byte Message (b) 64-byte Message (c) 64-byte Message using lddа/stda

	msg size $(byte)$	PIO inst.	O_s $(usec)$	B_s $(\frac{10^6 byte}{sec})$	O_r $(usec)$	B_r $(\frac{10^6 byte}{sec})$	$\frac{T_{round-trip}}{2}$ $(usec)$	L $(usec)$
(a)	16	ldd,std	0.6	26.6	3.0	5.3	5.5	1.9
(b)	64	ldd,std	1.4	45.7	11.3	5.7	14.6	1.8
(c)	64	ldda,stda	0.4	160	3.5	18.3	5.6	1.7

Table 1: Performance Characteristics of PIO Message Passing: (a) 16-byte Message (b) 64-byte Message (c) 64-byte Message using ldda/stda

represents the average over 100 trials.

Table 1 summarizes the extracted performance parameters. The results can be scaled to larger clusters because the user-perceived performance is not bound by Arctic's fat-tree network (320 MByte/sec/link, 0.15 usec/hop). However, these results will vary for different host architectures since the results directly reflect the host architecture's PIO capability.

A row-by-row comparison shows a significant improvement from the burst PIO optimization (row (c)). A 64-byte message can be transferred on UltraSPARC systems using burst PIO's at nearly the same cost as a 16-byte message (row (a)). Whereas, row (b) shows the dismal effect of accumulating single-beat PIO overheads on long 64-byte messages.

6.1.2 RDMA Mode Performance

Figure 6 shows StarT-X's RDMA bandwidth and latency as a function of transfer size from two experiments. Experiment (a) measures the time for S-bytes of data to ping-pong 100 times between two SMP's. Graph (a) plots the average one-way transfer latency ($L_{one-way}$) and the effective bandwidth ($S/L_{one-way}$). The peak RDMA bandwidth should only be limited by the efficiency of the host's memory system and its 32-bit 33 MHz PCI bus. We have observed the RDMA bandwidth of over 70 MByte/sec on Pentium PC's. Experiment (b) measures the time for two SMP's to exchange S-bytes of data simultaneously 100 times. Graph (b) plots the average time per exchange ($L_{exchange}$) and the effective bandwidth ($2S/L_{exchange}$). In an exchange, transmit and receive RDMA's are carried out simultaneously, but the aggregate bandwidth actually decreases due to PCI bus contentions. These experiments do not account for the software protocol overhead discussed in Section 5.3.

6.2 High-Level Interfaces

To support application development, two high-level communication interfaces have been implemented on top of the JAM primitives.

JAM Remote Procedure Call (RPC):

The JAM RPC library supports a simple non-blocking remote procedure call abstraction. The library is developed to support the runtime system of distributed Cilk[18], a multi-threaded dialect of C. The library contains two main primitives: JAM_rpc() to launch a procedure with an arbitrary-size argument buffer on a remote processor, and JAM_poll() to

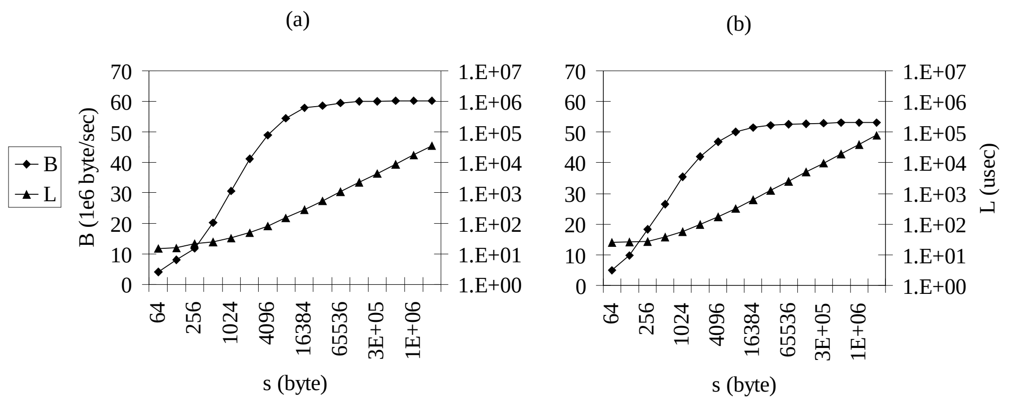

Figure 6: RDMA Bandwidth and Latency: (a) one-way (b) exchange

service the network and carry out pending procedure calls. The RPC interface supports multi-threading to allow parallel processes on an SMP to share a StarT-X NIU. The JAM_rpc() implementation automatically switches between PIO and RDMA modes according to the argument size.

MPI-StarT: MPI-StarT[11], a derivation of MPICH[8], has been developed to support the popular standardized programming interface. The MPICH channel interface is implemented using JAM primitives. MPI-StarT selects between PIO and RDMA modes to minimize transfer latency. An optimization allows same-SMP MPI processes to communicate at over 150 MByte/sec using SMP's shared memory.

Graphs (a) and (b) in Figure 7 plot the latency and effective bandwidth to transfer S bytes of data using JAM RPC and MPI-StarT. The measurements are conducted on the same hardware as the JAM kernel interface experiments. The experiment measures the time for S-bytes of data to ping-pong 100 times between two SMP's. The graphs plot the average one-way transfer latency L and the effective bandwidth (S/L). L for the smallest transfer size is 25.3 usec for JAM RPC and 40.5 usec for MPI-StarT. On large transfers, both JAM RPC and MPI-StarT approaches the peak StarT-X RDMA bandwidth on SUN E5000.

6.3 Applications on a StarT-X Cluster

MITMatlab[12] is a parallel numerical/linear-algebra package that supports the popular Matlab user interface. MITMatlab's parallel back-end uses MPI for communication and is easily portable to StarT-X by linking with MPI-StarT. We did not modify the MITMatlab source code for this demonstration. Table 2 summarizes the wall-clock time for multiplying two column-distributed single-precision floating-point $n \times n$ matrices using P processors. When testing on less than eight processors, we measure the performance both when the processors are on the same SMP and when the processors are divided between two SMP's. For $P = 16$ and $P = 24$, the experiments run on two and three 8-processor SUN E5000 SMP's respectively. Each SMP has only one StarT-X NIU. The results from the single SMP experiments are comparable to the results from MITMatlab linked with SUN's MPI library for shared memory. Our initial performance when using more than one SMP was poor because the stock MPICH broadcast implementation is oblivious to the difference between inter-SMP and intra-SMP communication bandwidth. However, after modifying the broadcast primitive, we have seen speedup on configurations of up to 24 processors over three SMP's. A comparison of rows corresponding to $P = x$, $P = x + x$ and $P = 2x$ shows that doubling the number of processors through StarT-X is as efficient as through the SMP memory bus. On two full SMP's ($P = 8{+}8$), linear speedup is still achieved on a $4k \times 4k$ matrix multiply. At three SMP's ($P = 8+8+8$) the speedup does drop to 14 percent below the expected linear speedup. Nevertheless, this level of performance and scalability is already well beyond what can be achieved on the same cluster using SUN's MPI library over 100 Mbit switched Ethernet.

7 Summary

This paper presents StarT-X, a 32-bit 33 MHz PCI NIU that supports user-level intra-cluster communication over the Arctic Switch Fabric. StarT-X and Arctic are currently installed on MIT/LCS's Xolas

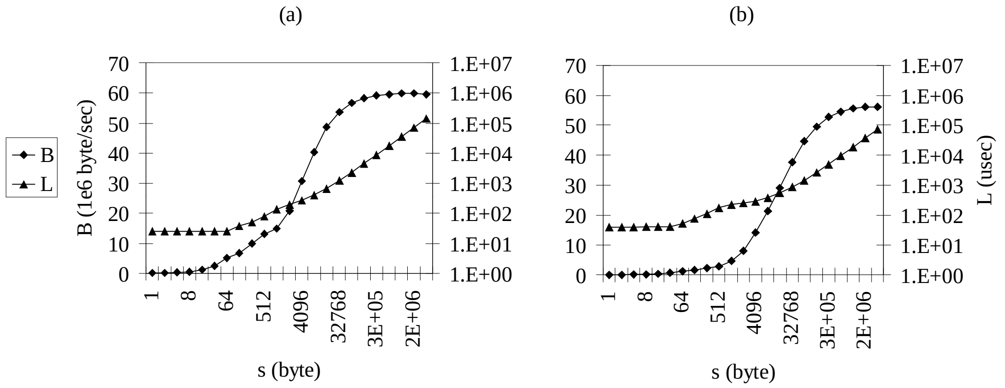

Figure 7: Bandwidth and Latency of High-level Communication Interfaces: (a) JAM RPC (b) MPI-StarT

	Matrix Size		
	1024×1024	2048×2048	4096×4096
	time (sec)	time (sec)	time (sec)
P=1+1	9.5	68.4	NA
P=2	9.4	68.3	NA
P=2+2	4.8	35.3	404.0
P=4	4.6	34.2	401.5
P=4+4	2.6	17.6	204.3
P=8	2.6	17.7	206.0
P=8+8	2.4	11.0	103.0
P=8+8+8	3.5	9.8	78.5

Table 2: Wall-clock Time for $n \times n$ Matrix Multiplication on P Processors. ("$P = 8 + 8 + 8$" means a total of 24 processors on 3 SMP's.)

Cluster of SUN E5000 SMP's. High-level programming interfaces (MPI and Cilk) and applications have been developed in a joint effort between research groups. At the moment, there is a plan to apply StarT-X and Arctic in a cluster of 32 Intel SMP's to support MITgcmUV[9], a global ocean simulation.

During this one-year StarT-X development, two lessons left a strong impression. First, VLSI technologies are improving so rapidly such that it is important to plan and track the technology changes even for a short 12-month project. When our project started, we had ruled out implementing a custom 64-bit or 66 MHz PCI interface because a 32-bit 33 MHz burst-capable master interface was barely feasible on FPGA's that had guaranteed availability at that time. A more recent survey has revealed FPGA's that would make us reconsider. Second, assuming one had a sufficiently fast interconnect, it is not difficult to build PCI NIU's that can operate beyond the limit of most hosts' I/O system by supporting burst PCI transactions. However, at the same time, these NIU's performance would still be sub-par in comparison to the rest of the system, i.e. processor and memory. This is either a reflection on the relegated importance of I/O performance in current architectures, or an indication that a truly high-performance NIU does not belong on the I/O bus.

8 Acknowledgments

This research is supervised by Professor Arvind of the MIT Laboratory for Computer Science. The author would like to give special thanks to P. Husbands for developing MPI-StarT and porting MITMatlab to StarT-X. The author also wants to acknowledge B. S. Ang, A. Boughton, D. Chiou, J. Costanza, M. Ehrlich, D. Rosenband, L. Rudolph, and X. Shen from the StarT-voyager team.

References

[1] D. S. Ang, D. Chiou, L. Rudolph, and Arvind. The StarT-Voyager parallel system. To appear

in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, October 1998.

[2] Applied Micro Circuits Corporation. *S5933 PCI Controller Data Book*, Spring 1996.

[3] 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*, February 1995.

[4] G. A. Boughton. Arctic routing chip. In *Proceedings of Hot Interconnects II*, August 1994.

[5] D. Chiou, B. S. Ang, Arvind, M. J. Beckerle, A. Boughton, R. Greiner, J. E. Hicks, and J. C. Hoe. StarT-NG: Delivering seamless parallel computing. In *Proceedings of EURO-PAR'95*, August 1995.

[6] D. E. Culler, L. T. Liu, R. P. Martin, and C. O. Yoshikawa. LogP: Performance assessment of fast network interfaces. *IEEE Micro*, Vol 16, February 1996.

[7] Dolphin Interconnect Solutions. *Multiprocessor Systems Design with SCI and Dolphin Technology*, 1995.

[8] W. Gropp and E. Lusk. User's guide for MPICH, a portable implementation of MPI. Technical Report ANL/MCS-TM-ANL-96/6, Argonne National Laboratory, 1996.

[9] C. Hill and J. Marshall. Application of a parallel Navier-Stokes model to ocean circulation. In *Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers*, pages 545–552, New York, 1995.

[10] J. C. Hoe and M. Ehrlich. StarT-Jr: A parallel system from commodity technology. In *Proceedings of the 7th Transputer/Occam International Conference*, November 1996.

[11] P. Husbands and J. C. Hoe. MPI-StarT: Delivering network performance to numerical applications. To appear in *Proceedings of SC'98*, November 1998.

[12] P. Husbands and C. L. Isbell. The parallel problems server: A client-server model for large scale scientific computation. In *Proceedings of the 3rd International Meeting of Vector and Parallel Processing*, 1998.

[13] M. Ibel, K. E. Schauser, C. J. Scheiman, and M. Weis. High-performance cluster computing using SCI. In *Proceedings of Hot Interconnects V*, August 1997.

[14] A. Krishnamurthy, K. E. Schauser, C. J. Scheiman, R. Y. Wang, D. E. Culler, and K. Yelick. Evaluation of architectural support for global address-based communication in large-scale parallel machines. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1996.

[15] Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*, 1.1 edition, June 1995.

[16] R. S. Nikhil, G. M. Papadopoulos, and Arvind. *T: A multithreaded massively parallel architecture. In *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992.

[17] PCI Special Interest Group. *PCI Local Bus Specification*, 2.1 edition, June 1995.

[18] Supercomputing Technology Group, MIT Laboratory for Computer Science. *Cilk-5.1 Reference Manual*, September 1997.

[19] Thinking Machines Corporation. *Connection Machine CM-5 Technical Summary*, November 1993.