# Gigabit Network with Cooperative Functions for General Purpose Massively Parallel OS

# 汎用超並列オペレーティングシステムと協調動作するギガビットネットワーク

by

Ryota Kunisawa

國澤 亮太

A Thesis

Submitted to

the Graduate School of
the University of Tokyo
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Information Science

Thesis Supervisor: Kei Hiraki 平木 敬
Title: Professor of Information Science

February 5, 1997

# Acknowledgments

I would like to especially thank Professor Kei Hiraki for helpful suggestions and checking the network interface card in fine detail. I also wish to thank Takashi Matsumoto for useful comments and discussions about user-level distributed shared memory architectures. In addition, I am grateful to other members of Hiraki laboratory.

## ABSTRACT

With the advance in performance of stand alone workstation and availability of network with high throughput communication channel, it becomes feasible to perform parallel applications on workstation clusters. However, existing network systems for workstation clusters offer multi-user, multi-job environment with software. Moreover, software is needed to construct communication data. Therefore, software overheads are fairly large.

A network system that reduces those overheads is proposed in this thesis. The network interface handles virtual environment by hardware, allowing users to communicate without software overheads. Moreover, by adopting communications based on framework of distributed shared memory, the overheads in message packing/unpacking is reduced, and the routing and traffic of message is administrated by OS. A prototype of this network is implemented. The effectiveness is evaluated using this prototype network.



## 論 文 要 旨

ワークステーション単体の性能向上と高速な伝送路を用いたネットワークの利用可能性により、ワークステーションクラスタにおける並列処理が現実となりつつある。しかし、ワークステーションクラスタにおける既存のネットワークはソフトウェアでマルチユーザ、マルチジョブに対応している。また通信データをソフトウェアで構築する必要がある。そのため、ソフトウェア処理のオーバーヘッドが非常に高い。

本研究では、これらのオーバヘッドを削減するための機構を持つネットワークを提案した。仮想化された環境を扱うためのハードウェアをネットワークインタフェイスに実装することで、ユーザはソフトウェアのオーバヘッドなしに通信ができる。また、 OS は共有メモリという枠組を用いた通信を採用することで、メッセージの組み立てに関するオーバヘッドを削減し、同時にメッセージの流路と流量を管理する。提案したネットワークのプロトタイプの実装を行ない、その性能評価により有効性の検証を行なう。

# Table of Contents

# List of Figures

# Chapter 1

# Introduction

The performance of stand alone workstation is rapidly increasing. However, users want more speedup in application execution time. Until now, massively parallel processing (MPP) systems are proven to be useful for running parallel applications. Therefore users want to run parallel applications on more commodity hardware, workstation clusters (or Network of Workstations, NOW).

However, traditional MPP system runs one application at a time and has dedicated network system. In contrast, NOW is a multi-user, multi-job environment and have poor network systems. Traditional OS doesn't have the ability to schedule distributed resources efficiently, and the increase of network performance, especially LAN performance, has not accompanied with the increase of stand alone workstation's performance.

We have proposed a network system which reduces latencies incurred at network interface controller, and cooperates with OS to offer parallel application execution environment for NOW. By providing an MMU and routing table to the network interface controller, remote memory access is done without software overheads and message traffics are reduced and controlled without software overheads.

Remainder of this thesis is as follows. In chapter 2, the required properties for NOW network systems for supporting massively parallel operating systems are described. In chapter 3, design and implementation of proposed network system is described. In chapter 4, the performance of a testbed of proposed network system is described. In chapter 5, the features as well as the problems of existing network systems and their features as well as problems are described. In chapter 6, conclusion and discussion are addressed.

# Chapter 2

# Required facilities for NOW interconnects

The main difference in communication between traditional MPP systems and NOW systems is that in NOW systems the user has no ability to use communication interface directly. In NOW, the OS manages the communication channel which are shared among multiple user-level processes. This results an OS interference in user-level communication, causing overheads.

The other difference is that the destination of a message is not determined statically on NOW environment. This is because the actual network topology is not known until the application is started, and processes may be scheduled on different workstations from previously scheduled workstations. The destination has to be resolved prior to sending a message, which is done by software.

In this thesis, this two requirement is met by providing an address translation MMU and network routing table into network interface controller. Those two facilities reduce both communication overheads and message traffics. Moreover, the OS controls message traffics by utilizing routing table. In this section, the importance of those two facilities are described.

## 2.1 Reducing Communication Overheads

In order to handle virtual environment without software overheads, the network interface has an MMU translates local process's virtual address into physical address. Therefore, when the NI controller receives a remote memory write request, it can directly write a received data into destination process's virtual address. When the NI controller sends a remote memory write request or receives a remote memory read message, it can directly read data from sender's or receivers virtual address. The routing table translates message destination into routing information of physical network. With this two abilities, the NI controller processes a network packet without the help of OS.

**Determine packet transmission route from virtual remote node ID.**

In multi-user, multi-job distributed shared memory environment, physical pages are moved from one node to other node when the process that uses those pages is re-scheduled on different

workstation. In order for a process to access moved pages seamlessly, the address of pages, which includes remote node ID, should be virtualized,

We call this virtualized address as Network Address. The network address consists of remote node ID, process ID, and virtual memory address of that process. Network systems for NOW should handle network address without OS interference.

In my network system, switching node determines the destination according to packet header. So, network interface card has a table which translates virtual node ID into routing information. If that ID doesn't hit the table, the card may interrupt the host and the translation can be handled by software or OS.

**Determine physical page address from given network address.**

As described above, network address also has process ID and virtual memory address. The network interface card should have an MMU, or at least TLB that translates virtual address into physical address. If the whole address translation table is too costly to implement in network interface, the interface may interrupt the OS and translation is done by OS or use DMA to access the page table constructed in workstation's main memory, when the TLB in network controller is missed.

The failure of address translation has a meaning that the actual page (related to virtual address) has not been loaded onto physical memory. If the corresponding physical page is swapped out to another node (which is very likely on NOW environment where network is faster then 2nd level storage), the remote memory access message should be forwarded to that node. If the physical page is swapped out to 2nd level storage, the network interface interrupts the OS, which then handles the message.

## 2.2 Reducing and Controlling Message Traffics

A multicast happens, for example, a page which obeys update protocol is shared by multiple node, and someone writes into this page. Multicast messages are reduced by controlling routing information table, which also stores page sharing information.

### 2.2.1 Multicast

In NOW environment, although each link has the same bandwidth, the available bandwidth between two nodes is not necessarily the same. For a simple example, a scalable network system often has hierarchical topology. In this case, the distance of two nodes gets farther and farther, the number of lower level nodes increases. Thus the available bandwidth in an upper level link per lower level nodes goes lower and lower, when an upper level link and an lower level has the same bandwidth. Therefore it is vital to reduce the communication between two distant nodes. When a communication is carried out by two nodes, those two nodes should be scheduled closely, because

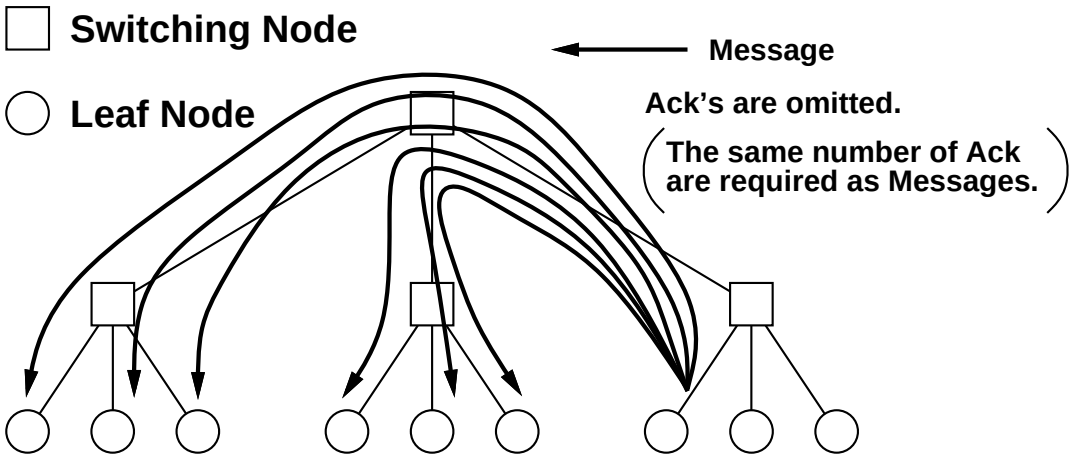


Figure 2.1: Simple Multicast using 1 to 1 communication

doing this will also reduce communication latencies incurred at switching nodes. A multicast, on the other hand, creates many communications between distant nodes in accordance with the number of multicast target nodes (Figure 2.2.1).

If a network system only supports one to one communication, a multicast simply requires the number of communications equal to the number of distant nodes[1]. When OS can find any kind of hierarchy in multicast message, a packet from upper node can be copied at switching node and sent to multiple lower nodes (Figure 2.2). This reduces required bandwidth of messages at intermediate links.

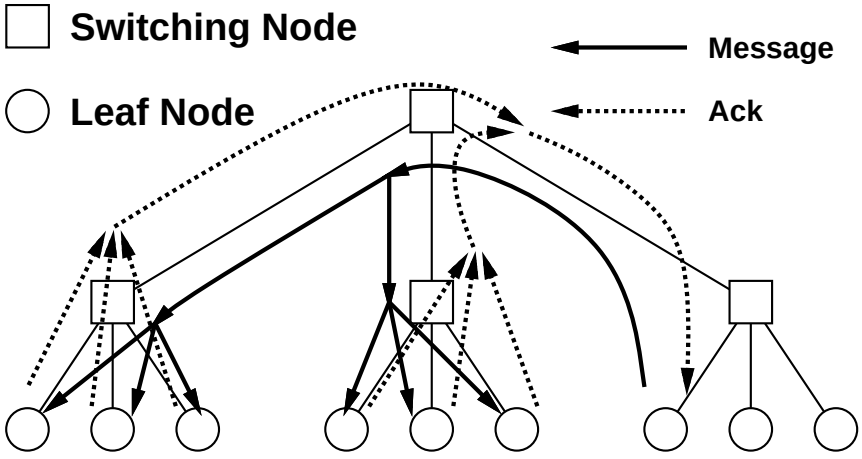


Figure 2.2: Copying Multicast

A reverse case of this is that packets can be merged into one packet if those packets have the same meaning. When a multicast message requires acknowledgment (ACK) from each nodes, those ACK's can be merged into one packet.

[1]Note, however, that it is not infeasible when a link is fast enough and there are a lot of bandwidth available (eg. any two end-nodes are connected directly each other.)

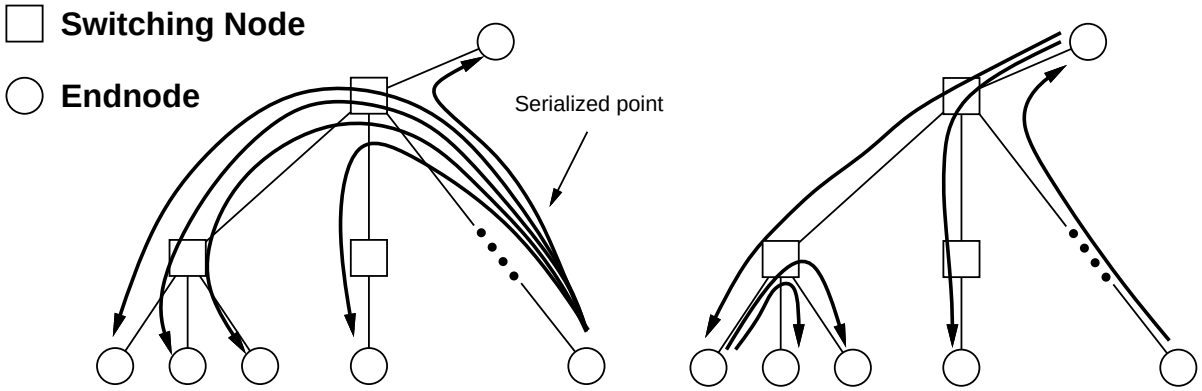


Figure 2.3: Packet can be forwarded and copied at end-node.

ACK combining at switching node is difficult in NOW environment, because the timing when a switching node receives ACK are different for each end-node and not all end-node will reply with ACK. In case that a switching node does not have these copying and combining capability, the interface card itself needs an ability to copy the packet or forward a received message to another node (Figure 2.3).

The latter is also useful for multicast. When the interface card receives a multicast message, it stores the message and forward it to one another receiver. This is called Chained Multicast (Figure 2.4).

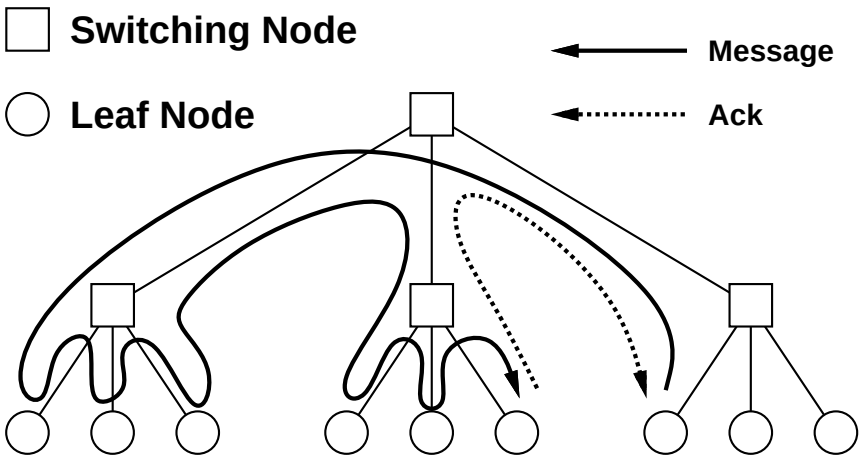


Figure 2.4: Chained Multicast

If a sender of multicast message requires acknowledgments from each receivers, a receiver adds it's own acknowledge to the end of multicast message and forward it to another receiver. The last receiver sends all acknowledges to the multicast sender in the end.

The ability to forward a packet to another node is also crucial to NOW environment, in which a process is sometimes re-scheduled to another node. A message may arrive a node, which is a valid destination node when the message is sent but no longer valid because the destination process is scheduled to different node while the message travels. In this case, the message should be forwarded

to a new valid destination node. If such messages can't be forwarded to valid destinations, moving a process requires synchronization of messages which incurs heavy overheads at context switching time.

The information about multicast message traverse is stored in routing table. In this way, the page directory (routing table in disguise) is distributed among several end-nodes. Therefore the software has complete freedom in implementing a page directory, and the header size of a multicast message is restricted in practical size even when a page is shared among enormous number of processors.

### 2.2.2 Atomic Remote Memory Access

Atomic remote memory access is an useful feature for synchronize a set of processes. When atomic remote memory access is not available as a basic operation, it is implemented as a sequence of properly synchronized remote memory read access and remote memory write access.

However, when workstation's I/O bus has the ability to access the data atomically, using this ability greatly reduces the overhead of issuing several remote memory access operations and saves network bandwidth. An intelligent NI controller which supports atomic remote memory access operations are feasible.

# Chapter 3

# Design and Implementation

## 3.1 Operating System

In order to fully utilize the proposed network system, a parallel operating system is needed which has the ability to schedules message traffics and page replacements efficiently.

We are developing a general purpose, massively parallel operating system, named *SSS–CORE*[17]. *SSS–CORE* uses a spin-wait which only spins when the requested resources seems to be to available soon. The likelihood of resource availability is determined by snooping resource usages presented by the OS. Thus the spin-wait is called Snoopy Spin wait (SS–wait). *SSS–CORE* and parallel applications run on the top of it use SS–wait for all basic synchronizations. In order to implement SS-wait efficiently, *SSS–CORE* needs distributed, shared memory at user-level.

Although *SSS–CORE*[18] does offer these capabilities with efficient software mechanism, there's no need to execute software when the proposed network system is used.

### *SSS–CORE*'s Remote Memory Access

*SSS–CORE* offers the ability to write other process's memory without overheads[18]. This is done by providing a process with an access ID. A processes which owns other process's access ID can access that process's memory. To get access ID of another process, a process have to provide it's own access ID.

We have also added this ability to our network system. Remote memory access message contains process ID and access ID of a process of which memory is accessed, as well as process ID and access ID of a process which initiates remote memory access.

When actual access ID of accessed process differs from that written in request message, the request message regarded as illegal. In this case, the network interface interrupts the OS, and the OS will take whatever action according to *SSS–CORE*'s policy. It is required that for user-level process not to falsify the access ID, the access ID is written into request header by the network interface. The OS stores this access ID to network interface controller at context switching time.

## 3.2 Network Systems and Hardware Level Packets

The switching network proposed in this thesis is designed to be scalable, and has following features.

- A link has high bandwidth.
- Any network topology is allowed.
- Messages are sent and received in FIFO order, if desired.
- Reliable. No messages are dropped. Errors are detected.

To achieve first three features, a packet header contains routing information, and a switching node incorporates virtual cut-through routing. As described in previous section, a switching node might process copying multicast and/or combining of ACK packets[1]. Therefore packet type should be stored at the very first of packet header.

In virtual cut-through, when a switching node can't forward a packet from arriving port to destination port, the node processes the packet in store-and-forward manner. Thus, proper flow control method is required. If flow control is performed during packet transmission, the controller has to use both sending port and arriving port of one link, and link bandwidth is reduced by half. (You can't send a packet and receive another packet simultaneously through one link.) Flow control should be performed with a packet as an unit of one flow.

In order not to suspend a packet transmission, the size of one packet size must have a limitation.

When the packet is a remote memory write request, the location of whole to-be-written data should be in range of one page. Otherwise the receiver have to watch the address against page boundary, and if the address will cross the the page boundary, it must check the next page's attributes. If the attribute is different from previous page (e.g. has different protection), the interface controller may perform different operations. This may result in rather non straightforward reply message, part of remote memory write has succeeded, while other part has failed.

Therefore, the size of data in a packet must not exceed a page size (which is 4KB as of this paper's implementation) and data must not cross page boundary.

## 3.3 Interface Node

The network interface card consists of data transceiver and interface controller. The function diagram of network interface card is depicted in Figure3.1.

When the interface controller received a send data request from host computer, it starts DMA to move data from main memory to transceiver. The data path between main memory and interface

[1] Whether a switching node copies multicast and combines ACK packets depends on how much user spends money on switching nodes. An intelligent switching node is more expansive than an unintelligent one. Anyway the OS knows about switching nodes and determines how packets should be handled.

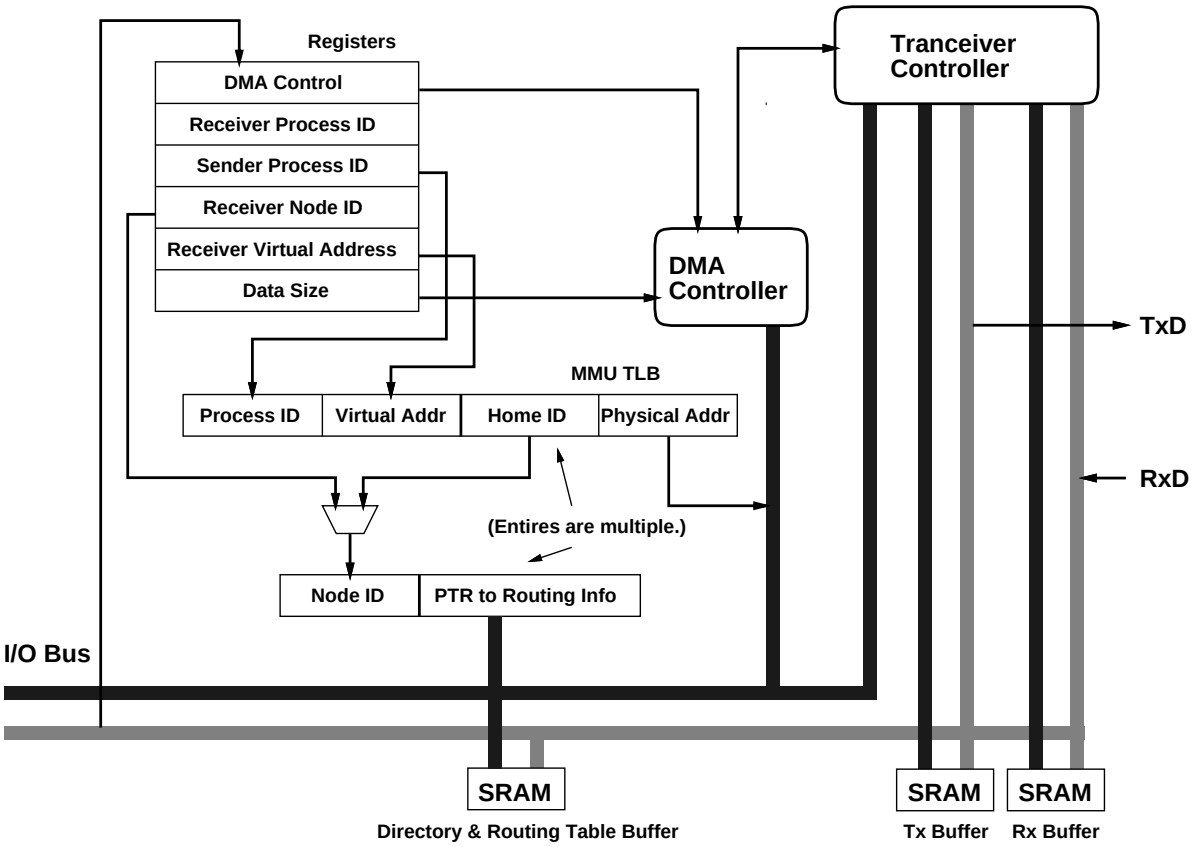


Figure 3.1: Function Diagram of Network Interface Controller

controller is usually not available at the time when the interface controller needs it. This is because the path is usually an I/O bus which is shared with other I/O cards. When the controller receives a first datum of a packet from transceiver, it stores the remaining, incoming data into on-board buffer so as not to drop the data while receiving. If the on-board buffer becomes nearly full, the interface controller sends a control message to stop the incoming data. When the buffer becomes to have sufficiently space, the controller sends a control message to incoming port to permit sending a packet.

Then the controller processes the packet according to it's type. If the type is unknown to the controller, it interrupts the host cpu and the packet is processed by software (OS).

The controller has an address translation unit which translates network address into physical address of local memory. Because the network address is 64bits, complete page table may not be constructed on network interface memory. Thus, the controller doesn't need to have page table locally. When TLB miss has occurred, it uses DMA to fetch a table entry corresponding to given network address from main memory, or interrupt the OS so that the OS will translate the network address. Note that the latter case, software is not executed unless the TLB misses.

Routing information is handled similarly. The network interface translates remote node ID into (variable sized) routing information. If the translation failed, the controller interrupt the OS, and OS will translate remote node ID into routing information.

### 3.3.1 Packet type and it's handling

The basic functions of network interface is fairly simple. According to previous section, a network interface controller needs following information to initiate remote memory access.

| Information | Brief explanation |
|---|---|
| Request Type | Packet type. Read/Write/Atomic |
| Network Address of Receiver | Node ID, Process ID, Virtual Address |
| Access ID of Receiver | For checking capability. |
| Data Size | Size of sending data (write), requested data (read) |
| Starting Address of Data | Start address of data |
| Process ID of Sender | Required for checking protection. OS writes this field. |
| Access ID of Sender | The same as above. |

#### 3.3.1.1 Remote Memory Write

**Sender**

A remote memory write operation is started when the host computer issues a write operation into network controller. The controller first checks the destination node ID of receiving process and translates it into routing information. Then the controller uses DMA to copy the data from main memory to it's buffer, constructing a packet. After then, it sends the data to output port.

**Receiver**

When a network interface controller receives the packet, it checks the protection according to access ID. If it succeeds, the network controller translates network address into physical memory address, and uses DMA to transfer data. If the address translation fails, the network interface controller interrupts the OS, and the OS handles the request. The network interface card also interrupts the OS when the protection check has failed.

#### 3.3.1.2 Remote Memory Read

Remote memory read is implemented as a reply of remote memory write from remote host.

**Receiver**

When a network controller receives a remote memory read request, it uses DMA to get the requested data from main memory, and send the data back to requester using remote memory write operation.

**Sender**

For the remote node to be able to write requester's local memory, the sender of remote memory read request put the remote address of requesting data and local address of requesting variable into request header.

#### 3.3.1.3 Multicast (Packet Forwarding and Coping)

The network interface card has an ability to forward the received message to another node in order to support chained multicast. If chained multicast is arrived at the network interface, it extracts the next (virtual) node ID from packet header, translates it into routing information, and send the packet to sending port. Message coping is handled as the same manner.

## 3.4 Switching Node

Because the routing information is stored in packets and the maximum size of a packet is limited, a simple switching node is sufficient. Packet coping and ACK combining node is optional.

# Chapter 4

# Evaluation

## 4.1 Testbed

A testbed of proposed network system is described here. The current testbed consists of only end-node network interface. No switching node is currently available.

### 4.1.1 Physical Link

For physical links, I used optical Fibre Channel transceiver.

Fibre Channel offers a high speed, serial data transmission link. It supports many physical transmission media, such as single/multi mode optic fiber, coaxial cable, and twisted pair, and support wide range of transmission speed from 533Mb/s to 4Gb/s. This standard is primary aimed to replace current I/O interface, so it defines point-to-point connection and star connection, and lately loop connection.

FCS (Fibre Channel standard) is an ANSI (X3T11) standard that defines a connectivity scheme. FCS is divided in several layers. FC-0 defines a physical transmission, such as media, cabling, and connectors. FC-1 defines transmission code and delimiters. FC-2 defines packet format and point-to-point protocol. Serial SCSI and Gigabit Ethernet use Fibre Channel as physical level link.

The transceiver used in this testbed is clocked at 53.125MHz, and has separate send port and receive port. Thus a link can send and receive at the same time. A port can transmit 20b data per clock. Each port has a bandwidth of 1.06Gb/s, so a link has total bandwidth of 2.12Gb/s. However, data are encoded when transferred using 8B10B[3] encoding (an 8bit data is encoded to a 10bit to be transfered), so actual bandwidth of data is 800Mb/s for a port (or 1600Mb/s for a link).

### 4.1.2 Network Interface Controller

An FPGA (Xilinx[4] XC4025E) is used for on-board controller, for developing and debugging with ease.

For on-board buffer, SRAM's (512K×8b) of which access time is 20nS (50MHz) are used. To

prepare a 32b bus between the interface controller and SRAM, four SRAM's are required. Thus the interface controller has a 2MB of buffer which throughput is 200MB/s.

The network interface card is designed as an SBus card. The card doesn't support 64b access mode of SBus. SBus is driven in 25MHz clock and can transfer 32b data in a clock. Maximum number of burst cycle supported by Sun SparcStation10/20 is 8, and each burst access needs at least 4 additional cycles for bus arbitration, the theoretical maximum throughput of this card is 67MB/s.

SBus also has it's own address space which is quite similar to that of Memory Channel[1]. So SBus has the same disadvantage as Memory Channel, in the sense that the user has to careful about excessive usage of SBus address space. In SunOS, SBus address space are usually only mapped (by the device driver, which resides in OS kernel) before the SBus card initiates DMA. When the SBus card ends the DMA, it interrupts the OS and the device driver unmap the allocated SBus address. If the allocation of SBus address space failed, the device driver queues an SBus address allocation request into kernel. The SBus card can not initiates DMA until the allocation succeeds, incurring latencies in DMA startup.

Although it is possible to implement a complete translation hardware from network address into physical address[2], I used SBus's address translation feature. Using the same value for process's virtual address and SBus address makes this possible, and the network interface controller can use process's virtual address directly. Using a readily available feature causes some limitation, however. The IOMMU can't handle as large address space as that of one process, the available shared-memory address space for a process is limited. And, because separate (virtual) SBus address space is offered to each process, the page tables of SBus address should be changed when a different process is scheduled[3].

## 4.2 Performance of Physical Link

The FCS optical transceiver can transmit raw 16b data (a doublet) for every 53.125MHz, thus it can achieve 106.25MB/s (850Mb/s) throughput. This transceiver requires roughly 10.1 seconds to initialize itself and synchronize it's receiver port so that received bit stream are correctly reconstructed as a doublet. Once initialized, the transceiver receive data at any time, as long as the received data is synchronized with the receiver's receive clock. To keep up the synchronization, the sender must send legal 20b code at every clock.

A 16b data is encoded to 20b by 8B10B method prior transmission. Because there are a lot

[1]To be precise, Memory Channel has a similar concept to SBus. In SBus, address translation hardware between SBus address and physical address is called IOMMU

[2]SBus can be accessed directly by physical address, without using IOMMU.

[3]The whole page tables need not be changed by limiting available SBus address for one process and assigning unused SBus address bits for part of process ID.

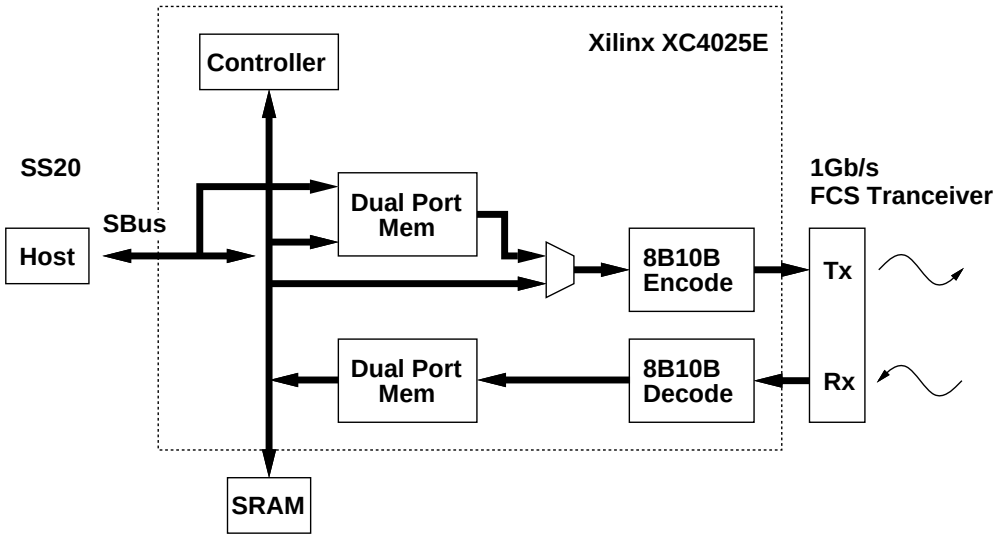


Figure 4.1: Function Diagram of Testbed Network Interface

of space unused in encoded 20b pattern, some of that unused 20b pattern are defined as control code[4]. Therefore a control code can be transmitted at any time while data is transmitted.

The latency of data transmission consists of the latency of transceiver and the time the data travels between the link, though the latter is negligible for links of short length (less than 10m). The transceiver used in this thesis has a latency of 50nS to send a doublet into receiver.

As mentioned above, a data must be encoded before transmitted, as well as be decoded after received. On this project, the encoder and decoder is assembled in (relatively slower) controller FPGA. Due to the time and space constraint of FPGA inner building blocks, the controller can encode doublet in 2 clocks of 53.125MHz, which limits the link bandwidth at 425Mb/s. (One clock duration of 53.125MHz roughly equals 18.8nS, and the encoder missed the constraint at about 2nS.) The latency of encoding a doublet is the number of pipe stages it requires which is 6 clocks. The decoding latency is 6 clocks, which happened to be same as the latency of encoding.

Therefore a link has latency of 163nS (50 + 18.8 * 6).

## 4.3 Overhead of Remote Memory Access

Because the interface controller directly access host workstation's I/O bus, the actual overhead is closely related to the overhead of protocols used by the I/O bus. Similarly, remote memory access may require page tables stored in main memory.

On this project, the interface controller is designed to hook into SBus[1] and used with Sun SparcStation 20 (SS20). SS20 drives SBus at 25MHz clock. As stated in previous section, this

[4] A 20b control code is represented with 'K' prefix (e.g. K28.5), whereas data code is represented with 'D' prefix (e.g. D28.5). FC-1 defines K28.5 D21.4 D21.5 D21.5 sequence as IDLE, which means that the link is idle and no data or control signals are transmitted.

project utilizes the SBus virtual address feature. So I need not implement TLB or page table access logic on the interface. SBus DMA access from network interface to main memory, when physical page is available and SBus address TLB is hit, is from 600nS to 800nS (+ additional burst clocks for burst access).

Thus the testbed requires $700 + 280 + 163 + 18.8 \times 2 \times 16 + 700 + 280 = 2723$ (nS) for transferring 64bytes.

# Chapter 5

# Related Works and Their Problems

Many researches have been done on reducing communication overheads on NOW environment. Those studies are introduced briefly, and their problems are addressed in later section.

## 5.1 Studies Based on Shared Memory Concept

### 5.1.1 Scalable Coherent Interface (SCI)

SCI[8] is an interconnect for tightly-coupled shared memory system. It's predecessor, Futurebus+, is a bus and therefore has a limitation in data throughput and scalability. SCI offers point-to-point link interfaces and handles up to 64K nodes. Although SCI systems are usually constructed as a ring topology, any topology can be constructed using SCI switches.

A physical level SCI packet contains target ID, source ID, packet type, and CRC. Depending on packet type, an SCI packet contains 48bit address. The interpretation of this address is up to SCI applications. SCI also supports cache coherent memory transaction. The consistency of cache are administrated by chained directory. A cache line in a processor has to have a pointer to previous and next cache line that caches the same address. The memory has pointers, for each cache line, to the first cache line in a processor which caches the corresponding line.

### 5.1.2 Memory Channel

Memory Channel (MC)[9] provides user-level, virtual-shared-memory facilities. MC defines it's own physical address space independently from each processor's virtual or physical address space. User-level application maps a page of MC's address space in `mmap()` manner, and the mapped page are attributed as uncachable. When the application writes into this page, the processors issues write request to processor bus. MC controller gets the request in turn, then constructs a write-request message and sends it to MC network. When constructing a message, MC controller adds a header which identifies the destination and tailer which is a CRC.

MC is currently implemented with Digital AlphaServer 8000 as end node workstation, and all end nodes are connected in a switch.

### 5.1.3 ServerNet

ServerNet[7] is a byte-serial (9bit command/data and 1bit clock), wormhole-routed, packet switched, point-to-point network. A transmitter defines the clock-rate. The end-node and switch has CRC-based error detection hardware. The physical link is ECL, but optic fiber is planned in future.

ServerNet provides common hardware and software services for both processor and I/O nodes. For communication between processors, ServerNet provides remote memory access without software execution at the remote node. In order to provide protection, the remote node CPU have to set permission bit in ServerNet network address translation table. The remote network interface card checks this table to translate network address into physical address and see if it can operate on this physical address. Remote memory operation provides Read Request, Read Response, Write Request, Write Response, and Unacknowledged Write.

### 5.1.4 Myrinet

Myrinet[6] is also a scalable, switching network. It's network topology is primary aimed at 2-dimensional mesh, but not limited to it. Originally, its physical transmission media is parallel wire cable. Optic fiber is used in near future.

It supports variable sized packet from two reasons.

1. The routing information is directly written in packet header. When a switching node gets a packet, it takes a port number from packet header and send the packet to the corresponding port of the switch. (Thus Myrinet supports only one-to-one communication.) The length of routing information equals to the number of switching node the packet hops.

2. The user can contain any size of data in a packet, thus can reduce some protocol overheads compared to the case when a very long data must be divided in small pieces and sent separately.

What makes Myrinet unique is that the network interface card has a RISC processor (called LANai). The user can program LANai so that it can take whatever action according to the contents of payload. Moreover, the user can map the data buffer of network interface to his process, and can reduce the overhead of copying packets.

## 5.2 Studies based on message passing concept

### 5.2.1 Active Message

The basic idea of Active Message[10] is that each message has the address of user-level handler in it's header, and the handler is called when the message arrives. The user-level handler should only retrieves data from incoming messages. This means that the computation phase and communication phase are separated. Parallel applications are scheduled at compile time to be executed

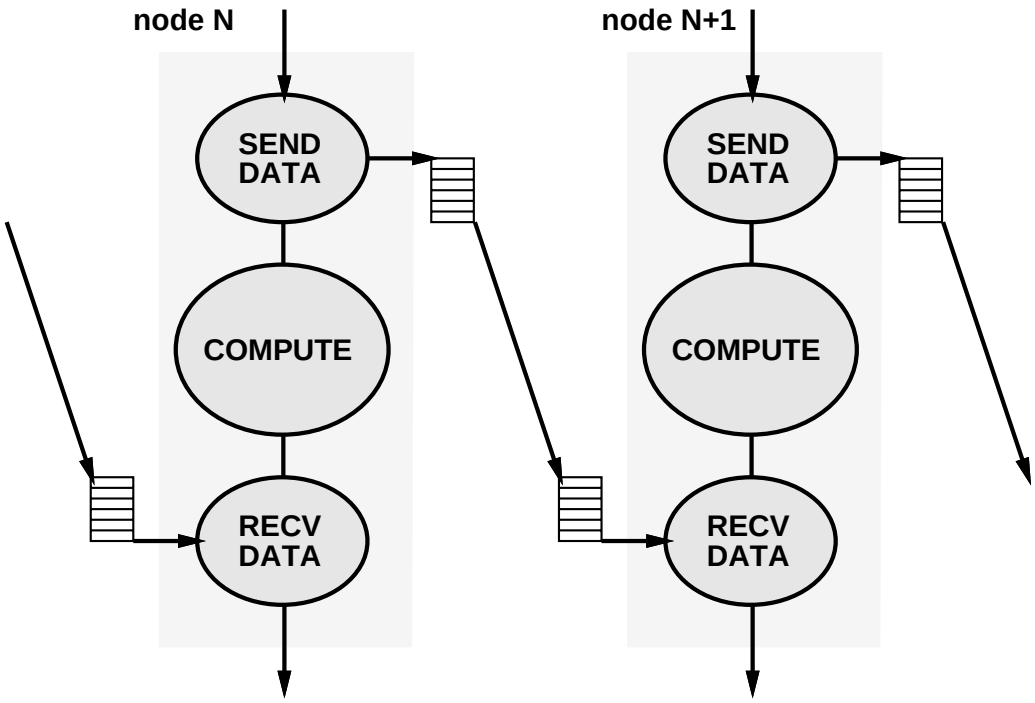


Figure 5.1: Active Message Pipelining

in pipeline manner. One stage of computation makes a data to be transferred, and the latency of data transmission should be equal to the time needed to execute one computation stage (see Figure 5.1).

Active Message is implemented on CM-5, and on ATM and SCI based workstation clusters[11][12].

### 5.2.2 Illinois Fast Message and Fast Message 2.0

Fast Message[14] is very similar to Active Message in that the message header contains the address of user-level handler. Because Fast Message targets not only MPP but also NOW, the user-level handler can not be called timely when the message arrives. Therefore, Fast Message needs message buffer, and user have to be careful about deadlock avoidance. (The whole system may be deadlocked when a buffer is not cleared for a long time.) Fast Message 2.0 supports streamed messages.

Currently Fast Message is implemented on the Cray T3D, and on Myrinet, ServerNet, and Memory Channel based workstation clusters.

## 5.3 Libraries

Application writers sometimes need standardized communication library for programming and porting applications with ease. For example, MPI[16] is such a library, and provides message send/receive interface which offers the following abilities.

1. Construct a message from a user-defined datatype.

2. Search messages which have the user-specified type from message queue.

Because libraries reside in one level higher layer then the studies described above, they are beyond the scope of this thesis.

## 5.4 Problems

The problems of existing network systems and protocols are described.

### 5.4.1 Send Processor's Request Directly to Network

Memory Channel and SCI send the processors read/write request directly to network.

Memory Channel focuses on physically closed network (on level cross-bar switching network) in which the number of processors are small (two to 96 processors). So that MC's address space is defined as 512MB (total number of page is 64K and page size is 8KB). If larger address space is defined as MC's address space, MC network can have more number of processors. In other words, to scale up MC network, one need to replace the MC controller with one which can address larger MC space. Moreover, because the address space of MC is limited and MC provides only single address space, the number of pages which can be mapped into all parallel applications are limited. The number of applications which request large MC address space is limited, which is not a pleasant on multi-job environment.

SCI needs a directory entry for every cache line at main memory.

### 5.4.2 Map Network Interface Buffer to User's Address Space to Reduce Coping Overhead

As for Myrinet, Myrinet Control Program directly maps Myrinet interface's buffer onto user's address space. In this case, if host creates sending packet directly on interface's buffer, the interface controller needs not copy the packet from somewhere when the packet is ready to send. However, because the amount of buffer on the interface controller is limited, the user should be careful about his usage of that limited buffer. If the user has allocated some part of the interface buffer but forgot about freeing this, there might no space left on interface buffer eventually. If one really has to handle this problem[1], we need some kind of virtual buffer at other places, possibly main memory and/or 2nd level storage. In this case, the buffer is swapped out when the physical buffer overflows, and swapped in when the user reads from or write to mapped buffer address. The interface card should better be hooked into memory bus than I/O bus when the I/O bus is slower than memory bus (and this is usually the case with). Otherwise, the system incurs extra overhead at swap time.

[1] I would here someone saying... "Well, even on Unix, one reason of `malloc()` failure is that there are no virtual memory left available. Why should we circumvent this problem in anyway?"

Even if the system doesn't provide the virtual buffer and there are no need of extra moving at swap time, there lies at least one implicit copying of data.

If remote memory access is not implemented at interface level (say hardware level) protocol, the software which creates interface level packets has to copy the data into the payload of sending interface level packets and copy the data from packets into destination address.

### 5.4.3 Active Message and Fast Message

Active Message[10] scheme works very well on dedicated parallel systems, where the latency of message transmission is statically analyzed, and there are no other processes running. However, it will not work well on multi-job environment in following reasons.

1. The original implementation only requires a limited message buffer at communication interface. In this case, user-level hander must not be interrupted while receiving a message, else it will happily drops incoming message. This leads unfairness in scheduling, when a user process send a very large data. Moreover, the handler should be fast enough to receive the incoming data stream. It's not really a drawback, because traditionally network speed is far slower than processor's data copy speed.

2. If one intends to eliminate the scheduling unfairness, the data have to be stored in buffer on network interface, and eventually copied to main memory. Then, the user-level hander, whose role was only to retrieve the data from communication link, no longer necessary.

3. Active Message scheme will not work on workstation environment at all. Because both the latency of message transmission and the time required for computation cannot be analyzed statically.

Active Messages implemented on the top of ATM[11] needs message buffers for each user-level process. Those buffers are allocated in user-level address space, but the physical pages are pinned down so that it won't be swapped out when the process is scheduled out. When an Active Message arrives from network, the kernel moves the data from arriving message slot info to corresponding process's buffer. When the process is scheduled, the handler of Active Message is called. To avoid message overflow, the sender polls the receiver and checks if any slot is empty .

In Active Messages on SCI[12], the sender queues the message to the receiver process's message queue, and the receiver polls it's local queue when it is scheduled. The sender refrains from sending another message until the remote destination queue becomes empty.

Fast Message is very similar to Active Message. However, Fast Message on multi-job environment needs a message buffer in case when the destination process is not scheduled. Therefore, the same problems as of Active Message also applied to Fast Message.

# Chapter 6

# Conclusion

In this thesis, we have designed a new network system for NOW environment. On NOW, parallel applications run in multi-user, multi-job environment. Therefore it is vital to support those virtual environment on hardware level.

To reduce communication overhead, the network interface has an address translation MMU which offers protected user-level communication with hardware level. To reduce and control message traffic, the network interface has an routing table. This routing table is also used as page directory.

In order to test the efficiency of proposed network system, a testbed network system has created, and performance of this testbed is evaluated.

# References


[1] Susan A. Mason: "SBus Handbook", *Sun Microsystems, Inc.* (1994)

[2] Tom Shanley and Don Anderson: "PCI System Architecture 3rd edition", *MindShare, Inc.* (1995)

[3] A. X. Widmer and P.A. Franaszek: "A DC Balanced, Partitioned Block, 8B10B Transmission Code", *IBM Journal of Research and Development*, Vol.27, No.5 (September 1983)

[4] "The Programmable Logic Data Book", *XILINX, Inc.* (1994)

[5] William J. Dally and John Poulton: "Transmitter Equalization for 4Gb/s Signalling", In *Proceedings of Hot Interconnects IV*, Palo Alto, 1996, Available from `ftp://ftp.ai.mit.edu/pub/cva/hot_i.ps.Z`

[6] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su: "Myrinet – A Gigabit-per-Second Local-Area Network", *IEEE-Micro*, Vol.15, No.1 pp.29–36 (February 1995)

[7] Robert W. Horst: "TNet: A Reliable System Area Network", *IEEE-Micro*, Vol.15, No.1 pp.37–45 (February 1995)

[8] "IEEE Standard for Scalable Coherent Interface (SCI)", IEEE Std 1596-1992

[9] Richard B. Gillett: "Memory Channel network for PCI", *IEEE-Micro*, Vol.16, No.1 pp.12–18 (February 1996), Available from `http://www.computer.org:80/pubs/micro/web/m1gil.pdf`

[10] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser: "Active Message: a Mechanism for Integrated Communication and Computation", In *Proceedings of the 19th International Symposium on Computer Architecture*, ACM, pp.256–266 (May 1992), Available from `http://www2.cs.cornell.edu/tve/ucb-papers/isca92.pdf`

[11] Maximilian Ibel, Klaus E. Schauser, Chris J. Scheiman, and Manfred Weis: "Low-Latency Communication Over ATM Network Using Active Message", *IEEE-Micro*, Vol.15, No.1 pp.46–53 (February 1995), Available from `http://www.cs.cornell.edu:80/Info/Projects/CAM/hoti-94.ps`

[12] Maximilian Ibel, Klaus E. Schauser, Chris J. Scheiman, and Manfred Weis: "Implementing Active Message and Split-C for SCI Clusters and Some Architectual Implications", Sixth

International Workshop on SCI-based Low-cost/High-performance Computing (SCIzzL-6), Santa Clara, CA (September 1996)

[13] R. Martin: "HPAM: An Active Message layer for a network of HP workstation" In *Proceedings of the IEEE Symposium on Hot Interconnects*, (August 1994), Available from `ftp://ftp.cs.berkeley.edu/ucb/CASTLE/Active_Messages/hotipaper.ps`

[14] Pakin, Karamcheti, and Chien: "Fast Messages (FM): Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors", To appear in *IEEE Parallel and Distributed Technology*, 1997, Available from `http://www-csag.cs.uiuc.edu/papers/fm-pdt.ps`

[15] Vijay Karamcheti and Andrew A. Chien: "A Comparison of architectual suport for messaging on the TMC CM-5 and the Cray T3D", In *Proceedings of the International Symposium on Computer Architecture*, June 1995, Available from `http://www-csag.cs.uiuc.edu/papers/cm5-t3d-messaging.ps`

[16] *Message-Passing Interface Forum, MPI: A Message Passing Interface Standard*, June 1995, `http://www.mcs.anl.gov/Papers/Lusk/mississippi/paper.html`

[17] T. Matsumoto, S. Huruso, and K. Hiraki: "General Purpose Massively Parallel Operating System *SSS–CORE*", In *Proceedings of 11th Japan Society for Software Science and Technology*, pp.13–16 (October 1994)

[18] T. Matsumoto, T. Komaarashi, S. Uzuhara, and K. Hiraki: "A General-Purpose Massively-Parallel Operating System : SSS–CORE — Implementation Methods for Network of Workstations —", *IPS Japan SIG Reports*, Vol.96, No.79, pp.115–120 (August 1996)