Architecture and evaluation of Maestro2 high speed cluster network

Shinichi Yamagiwa : INESC-ID Rua Alves Redol, nº 9, 1000-029 Lisboa PORTUGAL { yama@sips.inesc-id.pt }
Kevin Ferreira : PDM&FC Rua Latino Coelho 87, 1050, Lisboa PORTUGAL {kferreira@pdmfc.com, lcampos@pdmfc.com}
Keiichi Aoki, Masaaki Ono, Koichi Wada : Institute of information sciences and electronics, University of Tsukuba, Ibaraki 305-8573 JAPAN {k1@padc.mmpc.is.tsukuba.ac.jp, ono@is.tsukuba.ac.jp, wada@is.tsukuba.ac.jp}
Leonel Sousa : INESC-ID Rua Alves Redol, nº 9, 1000-029 Lisboa PORTUGAL { las@inesc-id.pt }

Abstract

Cluster computers have become the vehicle of choice to build high performance computing environments. To fully exploit the computing power of these environments, one must utilize high performance network and protocol technologies, since the communication patterns of parallel applications running on clusters require low latency and high throughput, not achievable by using wide- or local-area network technologies. This paper identifies the main drawbacks of conventional network technologies, and the techniques proposed to solve them within a new network solution called Maestro, in an original version and with an enhanced architecture called Maestro2. This paper describes the design and the implementation of Maestro2 technology and the novel techniques used by Maestro2 to extract maximum performance from the physical medium. Experimental results, clearly show that Maestro2 is a promising technology, presenting very good results both in terms of latency and throughput, using both specialized communications libraries (proprietary MMP or standardized MPI) and well known communications protocols (TCP/IP).

1. Introduction

It is widely accepted in the scientific community that some of the global problems faced by our planet today may only be solved by increasing our current computing capability[2][5]. Energy, medical science and molecular biology, earth science and climatology, cosmology, machine intelligence and robotics are just a handful of examples of areas and applications in which further development strongly depends on the magnitude of computational capacity made available to tackle them[6]. Cost-constrains have driven the high performance computing community from specialized supercomputers to the idea of physical clustering of computers, which led to the concept of commodity cluster computing. PC clusters and Beowulf-type systems[23] are the two commodity clusters designs most used for implementing high performance computing systems. A cluster can be characterized simply as a collection of computers interconnected by a network in a physically limited space, working together as a computer resource. Research performed during the last decades on parallel processing shown the importance of

interconnection bandwidth and latency, beyond the interest of shared memory models and the need for lightweight control software. It is known that in general, clusters do not excel in either of these issues. However, research efforts have been made by the parallel computing community in order to overcome these drawbacks[13]. These efforts include, among others, new scheduling approaches and algorithmic techniques[28], that take into account latency or are latency tolerant, and the development of Distributed Shared Memory (DSM) systems[14][16][7] with explicit synchronisation mechanisms. Even so, some problems and applications require the high bandwidth and low latency only found in the tightly coupled Massively Parallel Processing (MPP) Systems[26]. As a consequence, networks optimised for cluster computing have been developed during the last years[3][27], which discard for example, high-level communication guarantees, such as multiple error correction and layered connection quality maintenance, not required for this class of systems.

To use clusters as high performance computing systems, one must exploit to the limit of the capabilities of all components and eliminate the bottlenecks found in the overall system. There are subsystems in all communicating blocks, from the user parallel application to the network, that play a significant role in the overall system performance. It is not easy to clearly identify the direct effects of each one of these onto the performance of overall system, but in general, to achieve high performance, it is required to pay particular attention to the following issues: *i*) to design a parallel application that exploits the high performance characteristics of the system, the programmer must have a deep knowledge about the main features of the cluster system, and pay attention to their main bottlenecks; *ii*) the cluster middleware must be efficient, which means to use suitable algorithms to perform the parallelization of the tasks in an efficient manner; *iii*) the performance of the processing nodes, which can be PCs, workstations or multi processor, such as a symmetric massively parallel (SMP) nodes, must be at least high enough not to decrease the overall performance of the system, including its sub-systems such as processor(s), memory and disks, and the operating system must usually be optimized, in order to contribute to a good overall performance; i*v*) the communication system, including communication software, network cards and switch equipment, must allow fast exchange of information among nodes; cluster computing needs high throughput for sharing large amount of information and low latency to be efficient at changing short synchronization messages.

This paper addresses the problem of designing a high performance communication system for cluster computing, in order to improve the overall performance of these types of systems. Fast but general purpose network communication system are used in general, but they exhibit several drawbacks for cluster computing. One of these drawbacks is related with the fact that data is normally encapsulated with extra information used to increase reliability and security. For instance, Ethernet, is a well-known and one of the most used protocols for the lower communication layers. When small amounts of data are to be sent (4 bytes), the header information introduced by the Ethernet layer could reach up to 90% of all communicated data at the physical layer[4]. This overhead reduces the effective throughput and increases latency also, due to the time required to analyse and process the additional data. Data fragmentation is another typical behaviour of conventional network protocols. Data fragmentation means

that when the data to be sent is larger than the used protocol maximum data size, data is separated in chunks and each resulting chunk is individually sent. This procedure reduces performance, as each chunk requires overhead information and the associated processing, and usually it implies to get individual access to physical medium before start sending each data chunk. Another drawback of traditional networks for cluster computing is the postponing of message transmission until the whole message is prepared by the link layer. Conventional link layer hardware sends a message only after the entire message is written into the buffer for transmission. This serializes the communication processes and degrades throughput. Finally, conventional networks use *hubs* or *switches* to interconnect computer nodes and these points typically form a bottleneck where data congestion occurs.

All these drawbacks associated with conventional network technologies can be seen as unnecessary overheads that degrade communication performance in particular and overall performance of cluster systems in general. If these overheads are reduced or even eliminated, it is possible to significantly increase communication and overall system performance. Even using some of the fastest conventional network equipment, such as Gigabit Ethernet[25], the problems mentioned above still occur. Although these networks could achieve high throughputs, the internal protocol and technology lead to high latency values, not suitable for high performance cluster computing.

Maestro cluster network[33][36] was developed to overcome performance disparities between the obtainable communication performance using conventional WAN or LAN based network technologies and the requirements of cluster applications. A very important issue is how to achieve two different but complementary aspects of the communication, low latency and high throughput. The former must be employed for frequent inter-processors synchronization, while the latter is fundamental for large data transfers.

Two main issues to improve network performance were identified and addressed in the Maestro cluster network, which led to two key techniques being developed, namely, *network burst* and *pipelined transfer*.

As a follow-on research to the Maestro project, we have been developing the next generation of Maestro network, called Maestro2[1][34]. This paper presents some of the new techniques used in Maestro2 to further increase performance, namely, *continuous network burst* and *out-of-order switching*.

This paper is organized as follows. Section 2 describes the Maestro network, stressing the main innovative aspects of the technology regarding to the other network technologies already available for cluster computing. Section 3 is focused on the implementation aspects of the network and in section 4 the experimental results obtained are presented. Section 5 concludes the paper.

2. Design of a cluster network

2.1. Background and definitions

Conventional cluster computers are usually built from off-the-shelf components. In spite of the clear advantages of building clusters using these types of components, such as low price and simplicity, there are serious disadvantages, especially when using conventional networking technology, like Fast Ethernet or Gigabit Ethernet along with TCP/IP.

The focus of this paper is on the cluster communication system, which is composed mainly by three distinct blocks: *i*) communication software; *ii*) network interface; and *iii*) network switching equipment. The communication system can experience bottlenecks both within the identified blocks, and, at the several interfaces they share with each other, with the kernel and with the user parallel application (through the cluster middleware, if existing).

In order to be able to clearly identify the main obstacles to achieve low latency and high throughput in a communication system, we considered the existence of four communication layers: 1) *Communication protocol software*; part of the identified communication software block, and responsible for the interface between the user program and the kernel or/and the network device memory; 2) *device handlers*; also part of the communication software block and responsible to perform the communication between user address space and device communication memory (this layer is usually located inside of the kernel as a form of device driver/module); 3) *data link layer;* this layer exists both at the network interface block and at the switch block, and it is responsible for managing the messages and preparing them for the physical layer, by doing operations such as framing, adding its routing information (headers), aggregating packets and performing DMA invocations; 4) *physical link layer*; also present in both the network interface and switch blocks, and responsible for the lowest level of the communication system. The following paragraphs will discuss the main bottlenecks for each of the four identified layers.

1. Communication Protocol Software

The main problem at this layer is the repetition of data-copying operations before messages have been transferred or made available to their receiving applications. In general, each message must be copied between user and kernel memory space, and may be further copied between kernel space and the network device memory. Such copying operations increase communication latency. In addition, the user-to-kernel context switches can be seen as an additional overhead, especially in case of frequent inter-processors synchronization. This problem has been well addressed in [17], [18] and [31] communication libraries. They have implemented a zero-copy communication method that allows the underlying network devices to access messages directly at user memory space, thus bypassing kernel intervention.

2. Device Handlers

Virtual-to-physical address translation is an obstacle to bypassing kernel intervention. Since communication buffers at user level are allocated in the virtual address space, their physical addresses must be identified prior to the actual transfers by the underlying network devices. If those devices maintain the page tables and the TLB, (i.e. Translation Look-aside Buffer), they can take charge of such translation and therefore transfer user messages without kernel support. This also frees more CPU time for user applications and hides communication overhead further.

3. Data Link Layer

The first problem is the conventional message framing. If the protocol's minimal data unit size is too large, the network latency increases, in relative terms, when transferring short messages. On the other hand, if the maximum data unit size is too small, longer messages must be sent using multiple data units, which increases the framing operations and thus degrades the total throughput. The Ethernet data unit (frame), for instance, must include 36-byte link-to-link routing information and 46byte data body, which are in turn the minimum data unit size[4]. The maximum data unit size (frame) on the other hand is restricted to 1500 bytes, so as not to be able to monopolize a slow communication link. However, assuming that cluster nodes are located in a geographically closed environment, messages can be sent directly to their destination using shorter routing information. Hence, a new message data unit, suitable to cluster communications, should be designed rather than using the Ethernet data unit.

The second problem is excessive DMA invocations. Most network devices always invoke DMA transfers to send data over the network, whether or not the data size is too small to use DMA efficiently. This is regarded as excessive overhead especially when fine-grained communication is repeated for frequent inter-processors synchronization. Therefore, network devices should include another type of high-speed transfer logic for small amounts of data, and switch between this logic and DMA according to the data size.

The third problem is the transfer scheme currently used to transmit data through the physical link. Provided a packet is the smallest transfer unit dealt at the data link layer, most network devices do not aggregate independent packets into one, whether or not those packets are too small. In other words, they repeat an entire transfer set-up operation including physical link arbitration for each small packet. This causes the physical link to become idle frequently, and consequently degrades network throughput. To address this problem, network devices must be capable of packet aggregation.

4. Physical Link Layer

Multicast takes an important role in the master-slave or client-server model. Conventional multicast either broadcasts a message to all participating nodes, and has non-intended receivers discard it, or sends a copy of the message to each receiver one after another. The former is typical in carrier-sensitive media such as Ethernet, with the drawback of unnecessary message elimination by non-intended receivers. The latter is easy to implement, but incurs in excessive message-copying operations. Thus, the physical link layer should facilitate a multicast scheme that involves neither unnecessary discarding nor copying of messages.

Extensive research work has been done at both the device handler layer, to address virtual-to-physical address translation, like Myrinet and U-net[8], and at the communication protocol layer like GM[21] and PM. Maestro cluster network was first developed to address the performance bottlenecks at data link layer, and physical layers.

The first Maestro network offered two novel optimization techniques called *Network burst* and *Pipelined*

Transfer, which are described below:

1. *Network burst*

   By partitioning each message into fine-grained packets, we can better manage the buffers at both the receiver's side and sender's side. Network Burst aggregates multiple packets at the sender in order to best match the space currently made available at the receiver's buffer, and transfers them in burst to the receiver. The finer the packet size, the better buffer utilization is possible. This packet aggregation (in this context we call the aggregated packets *frame*) disregards message boundary. This means that one frame may include packets from multiple messages or that different frames contain one single message. The burst transfer of aggregate data units reduces the number of physical link arbitrations, each followed by miscellaneous device set-up operations.

2. *Pipelined Transfer*

   Conventional network devices do not simultaneously process two or more messages. In other words, when such devices accept a new message from their hosts, any subsequent messages are blocked until the device completes transmitting the current message onto the physical link. This causes the serialization of the following three stages of message transfer: (1) from the host memory to physical link at a sender side, (2) through the physical link, and (3) from the physical link to the host memory at the receiver side.

   To avoid this serialization, we again partition each message into fine-grained packets, and have each device component transferring packets in a pipelined manner. As soon as the sender's network device transmits a packet onto the physical link, it picks up the following packet from its host memory. Similarly, the receiver's device keeps delivering packets to the host as it receives each packet from the physical link. Therefore, the three serialized transfer stages are simultaneously processed in a pipelined fashion, reducing network latency. The finer the packet size and more device components participating in the transfer of packets, the more pipelined messages are possible. This also means that the technique improves transfer throughput.

A novel link protocol, dubbed *Maestro Link Protocol* was developed, which implemented the above optimization techniques. The protocol assumes a point-to-point half-duplex link between network devices. The host interface is implemented with a pair of sending and receiving FIFO buffers that exchange messages with the host machine.

While the sender host keeps passing messages to the outbound FIFO buffer, those messages are pipelined and partitioned one after another into finer-grained packets, which are then aggregated into a frame as large as the receiver's inbound FIFO buffer can store, and are thereafter transferred onto the physical link in a *Network Burst*. On the receiver side, those packets are reassembled into the original messages that are finally made readable from the receiving FIFO to the host. Many-to-many host communication needs the presence of a switch that establishes a Maestro-Link-Protocol connection with each host and routes a message from the source to the final destination host. To facilitate both *Network Burst* and pipelined packet transfer, the protocol provides the following three features: *i) FIFO flow control*; the

protocol defines a status register that indicates how many bytes the host can write to and read from its FIFO buffers. This scheme maximizes the message-spooling capability, prompts the host to exchange the next available message quicker, and therefore mitigates network latency. This advantage is further exploited by having two or more pairs of FIFO buffers; *ii) Fair link arbitration*; this feature is the key to realize pseudo full-duplex transfer between two end points of half duplex link. Maestro Link Protocol requests the current sender to release the physical link every *Network Burst* transfer, and immediately grants the receiver to use the link; *iii) Physical link flow control*; the *Network Burst* transfer needs to know *a priori* the FIFO capacity currently available at the receiver side. This requirement is satisfied by carrying such capacity information in each frame. This is so, because the fair link arbitration feature guarantees two end points to exchange a frame with each other alternately. This capacity information is called *credit,* and represents the number of transferable packets[15].

A Maestro cluster network[33][36] is composed of network interfaces and switch boxes. Network interface is connected to the host processor through the 32-bit PCI bus at 33 MHz and exchange messages with host processor. A message is a communication unit composed of one or more packets. Each switch box is connected to network interfaces via IEEE1394[12] cables and it's responsible for switching messages from the different network interfaces. The Network interface (NI), consists of five hardware components, namely: a PCI interface, an NI manager, two FIFO buffers, an Maestro Link Controller (MLC) and IEEE1394PHY. The NI manager is comprised of a PowerPC603e@200MHz and 64 Mbyte EDO DRAM. The Switch Box (SW) consists of six hardware components, namely four SB interfaces, an SB manager, and a switch controller. The SB interface includes two sets of IEEE1394PHY, MLC, and two FIFO buffers. The SB manager consists of a PowerPC603e@200Mhz and 64Mbyte EDO DDRAM.

Although Maestro cluster network was able to achieve low latency and high throughput, during the development of the network, two additional issues that restricted the overall performance of Maestro cluster networks were identified. The first of these issues occurs at the link layer and it is related with extending the continuous transfer of data. The second issue occurs in the switch itself and it is related with reducing idle time. Two new techniques have been devised and their implementation led to a new architecture, named, Maestro2. The first technique is called *continuous network burst*. This technique is based on the idea of not terminating the burst transfer as long as there are one or more packets to be sent. The second technique is called *out-of-order switching*. In the out of order switching, the switch processes subsequent messages from any source when, due to congestion, it can not continue to transfer a given message. This technique reduces idle time of the switch buses and increases throughput.

2.2. New Techniques

2.2.1. Continuous network burst

To avoid the overhead in the link layer of cluster, we proposed a network burst in [33]. If the size of the message is smaller than the one of the transfer unit, the conventional transfer will only send the message and never aggregates it with other transfer units. In addition, the arbitration operations, which are needed for the sender to acquire the physical medium, are inserted among each transfer (Figure 1(1)).

The accumulation of arbitration time makes the utilization of the physical medium low, and thus the throughput will degrade.

On the other hand, the network burst transfer avoids the arbitration overhead with aggregating multiple packets into a transfer and sends it in burst to the receiver (Figure 1(2)). This will reduce the arbitration overhead, the utilization of the physical medium will be improved and thus the higher throughput will be achieved than the conventional transfer.

The network burst transfer is an effective method to improve the throughput. However, the length of a burst transfer is decided just at the beginning of the transfer. After the transfer starts, the sender never checks the capacity of its buffer and bursts the amount of data that is decided before the burst transfer starts. Assume that the sender's buffer is filled with data that is going to be sent during the burst transfer, the burst transfer will be terminated when the sender sends the amount of data that is decided at the beginning of the burst transfer even if there is more data that can be sent continuously. This causes the overhead to acquire the physical medium for the next transmission. The overhead will accumulate and finally will be a cause to degrade the communication performance.

To address this situation, we propose a new method to continue the transfer, called *continuous network burst*. First, continuous network burst will transfer an amount of data that is decided at the beginning of the transfer. When the burst transfer of the first burst transfer is going to complete, the sender will check the amount again without terminating the burst, if there are more data to be transmitted, inserts a *continue command* that notifies the continuation of the burst transfer to the receiver and continues the burst transfer until it detects the buffer is empty (Figure 1(3)).

As long as the sender's buffer is not empty and the receiver can receive, the continuous network burst will eliminate the arbitration overhead and the utilization of the physical medium will be improved. Thus, the higher throughput of the network will be achieved.

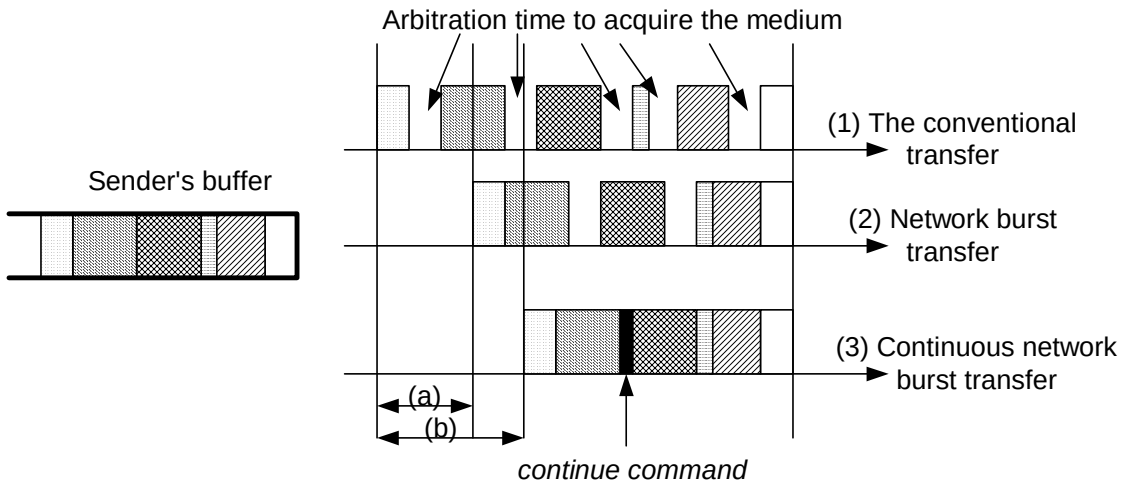


Figure 1 Comparison among the conventional transfer, network burst and continuous network burst

2.2.2. Out of order switching

Switch in a network transfers messages from the source port to the destination. This mechanism is done

based on a size the network protocol defined. For example, Ethernet switch transfers based on *Ethernet frame* size (i.e 1500 bytes) from the source port to the destination(s). This mechanism is the minimum function of network switch.

When a network switch is going to transfer a message from the source to the destination(s), the switching logic migrates messages from the source buffer to the destination buffer(s). During this data migration, the switching logic must control the data flow of the source and destination ports. The buffers could over- or underflow if the switching logic never checks the capacities of buffers. Finally, the migration operation will stop when the source or destination buffer becomes empty or full, respectively.

Here, let us focus on the ordering to process the message transfers on the switch logic. The switch logic stops transferring a message from the source buffer to the destination one in the case the source buffer is empty or the destination buffer is full. Conventional algorithm in the switch waits to transfer the message until the source or the destination buffer(s) have the capacity to provide/accept the subsequent data of the message. While this wait occurs, the data transfer stops and the switch device is idle. This means that the transfer operation in the network stops and the subsequent transfer operations are delayed until the current transfer completes, thus, the latency and the throughput of the network become worse. We call the mechanism *in-order switching* due to its strict ordering of transfers.

We propose a new technique to schedule transfers dynamically on switch logic, called *out-of-order switching*. In the case of the delay caused by the transfer wait time mentioned above, out-of-order switching picks up another message and tries to transfer it while the switch meets the idle time to wait for the buffer capacity. When the buffers' capacities change, the switch logic tries to resume the previous suspended transfer. This mechanism is similar to the Tomasulo's algorithm [9] in the dynamic process-scheduling field. On the cluster network field, we can expect this best effort effect to utilize the network's full bandwidth due to the suspend/resume mechanism for message transfers.

2.3. MLX network protocol

We propose a new network protocol for a high performance network, called MLX protocol. The connection of MLX protocol is configured by a point-to-point connection with full duplex medium as shown in Figure 2. Each side of MLX protocol controls its network buffer for sending and receiving data from/to other side. The network buffer is a FIFO buffer, which has *channel*(s). The channel is an independent data buffer to send and receive data. MLX protocol controls to send/receive data from/to each channel with respect to the sender's channel.

The flow control of MLX protocol performs credit-based. In the credit-based flow control, the sender receives a credit that is sent from the receiver when the receiver detects the change of its buffer. The sender can know how much data to send to the receiver from the amount of credit.

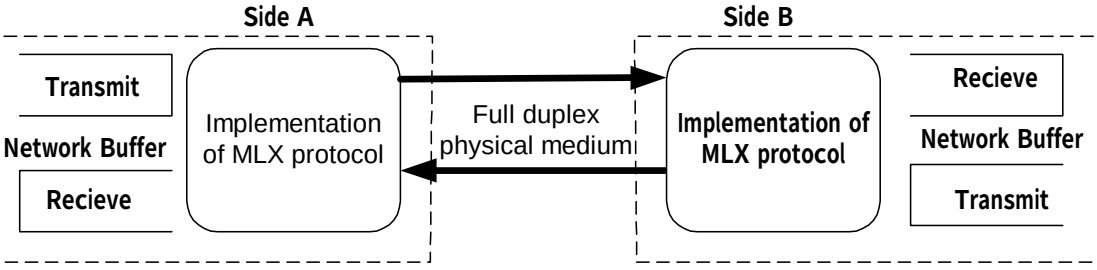


Figure 2 Configuration of MLX protocol

The MLX protocol sends and receives in a data unit with the full duplex physical medium. We call the data unit a *frame*. There are two categories of frame: *data frame* and *info frame*. The data frame includes: packets in the network buffer, a frame header and a checksum. The frame header includes a credit update information, continue command(s) and an acknowledge for the last receiving of the message. The info frame consists of a frame header of a data frame. Figure 3 shows the communication example of MLX protocol. The sender of a data frame must wait the acknowledge with respect to the channel until the next data transmission. In the figure, the side A receives an acknowledge (F1') of the data frame (F1). After F1', the side A can send the next frame with respect to the channel of the frame (F1). As well as the case above, the side B can send a data frame(F2) because it has not sent a data frame while the side A is sending the frame F1. While this rendezvous transaction occurs in one channel, the other channel can be utilized to transfer another data frames. Thus, the transmissions between channels make the best effort, to achieve high utilization ratio of medium. On the other hand, the capacity of network buffer is managed based on credit information when the capacity of the receiving buffer has changed. In the Figure 3, credit update information are sent to the side A as soon as the packets in the receive channel of the network buffer in the side B has been read. Credit update information includes the change of the receiver's buffer capacity. Due to the full duplex connection of MLX protocol, the credit update information from one side can be sent while the other side of the link is sending a frame. This makes the data frame transfer and the credit update information overlapped and thus the latency of the acknowledge will be overshadowed by the subsequent data frame transmission.

Because the MLX protocol is designed based on the Maestro Link Protocol on Maestro network, it also benefits the effect of the pipelined transfer technique. The Maestro Link Protocol is aimed to a half duplex link and exchanges the frames over the link with sharing the time. On the other hand, MLX protocol uses a full duplex link and can overlap the transfers of frame between two sides. This means that higher throughput is achieved with the pipelined transfer, while the latency will be hidden by the full duplex communication and thus the latency will be smaller than the Maestro Link Protocol.

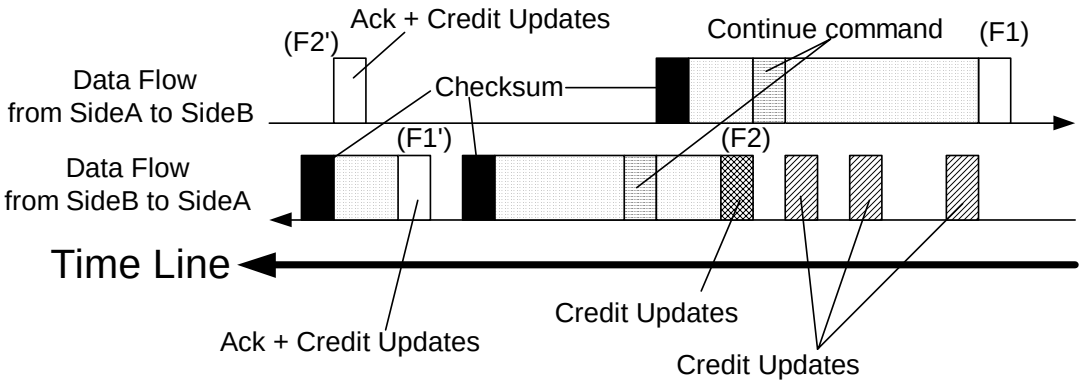


Figure 3 Communication example of MLX protocol

3. Maestro2 Implementation

3.1. Network hardware

The organization of Maestro2 network is similar to Maestro network, that is, network interface(NI) that is connected to a host computer via 64bit at 66MHz PCI bus and switch box(SB) that gathers connections from NIs as shown in Figure 4.

The network interface consists of Xilinx Virtex-II FPGA[32], 300MHz PowerPC603[20], LVDS ser/des chips[11] and 64Mbyte SDRAM. The PowerPC controls components on NI and implement the communication protocol among NIs. LVDS ser/des chips transfers data by 3.2Gbps and are connected to SB with LVDS's parallel cable. The SDRAM is a memory to maintain the firmware for PowerPC and data structures to save the states of messages and the requests from its host machine. The FPGA includes PCI bus interface, MLX protocol module that controls LVDS connection, a PowerPC interface and a DMA engine. The PCI bus interface provides an interface to transfer data between SDRAM, network buffer of MLX protocol module and host machine's memory. The PowerPC can access to the host memory via the PCI interface.

SB consists of eight MLX protocol modules, four message analyzers, a 300MHz PowerPC603 and a switch controller. Each MLX protocol module is connected to NI via LVDS cables and exchanges message. Message analyzer recognizes the message format in Figure 6 that includes *packets*, extracts an 8byte message header from each message that is received by the related MLX protocol module and stores it into its message header buffer. In our implementation, the packet size is 32byte. The PowerPC picks up the message header, decides the destination of the message by its routing firmware and writes a switching request to the switch controller. The switch controller analyzes the switching request and tries to execute a message transfer via the switching bus.

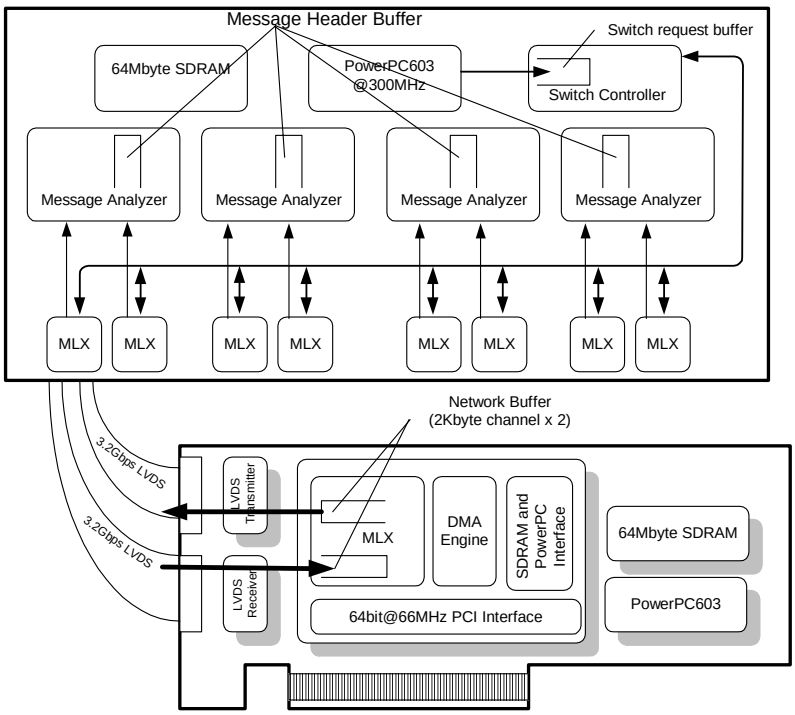


Figure 4 Organization of Maestro2 network

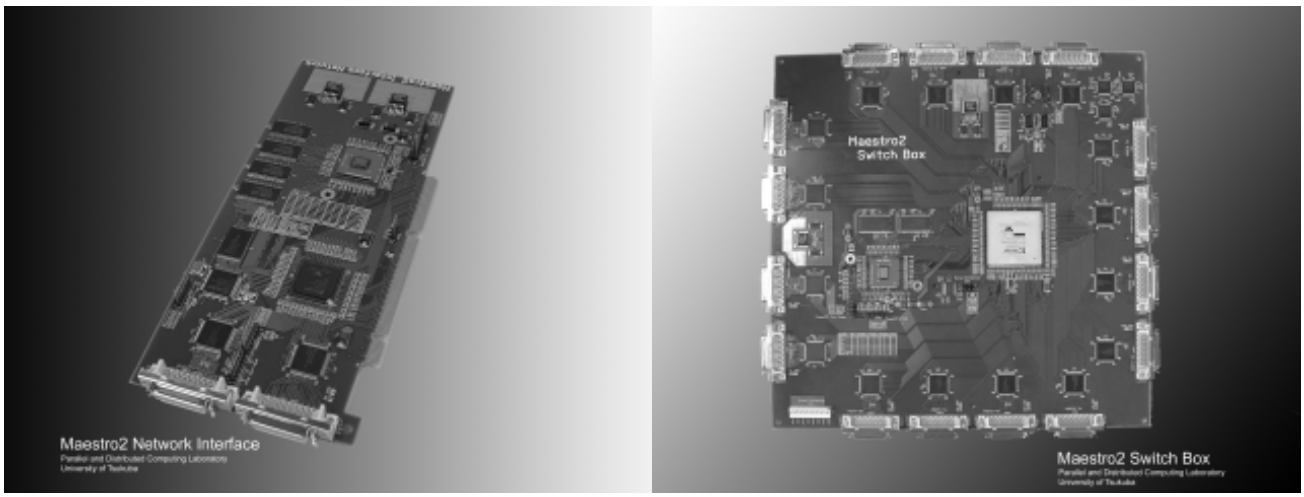


Figure 5 Maestro2 cluster network

Figure 5 shows the photos of Maestro2 cluster network's network interface card and switch box.

As shown in Figure 7, the switch controller is organized with three units: a request fetch unit, a reservation station unit and a transfer unit. The request fetch unit detects new requests in the switch request buffer and tries to store it into a reservation stations. The reservation station unit has one or more stations to store the switching request(s). The reservation station unit accepts, if an empty station exists, the switching request, in which the fetch unit is trying to pass data to the reservation station unit. The transfer unit will activate a request and process the transfer via the switching bus. When the unit detects the full or empty states of the network buffers, the transfer unit stops and the message transfer and thus tries to chose the subsequent switch request.

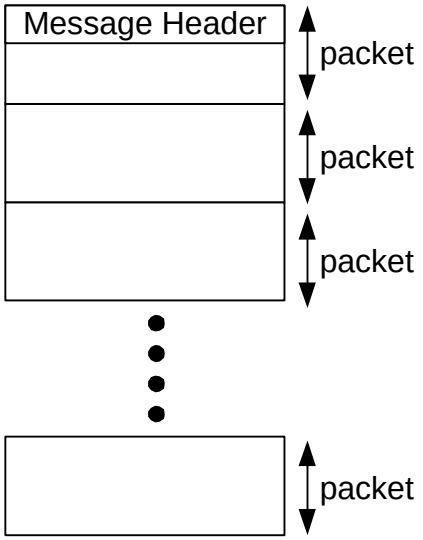


Figure 6 Message format

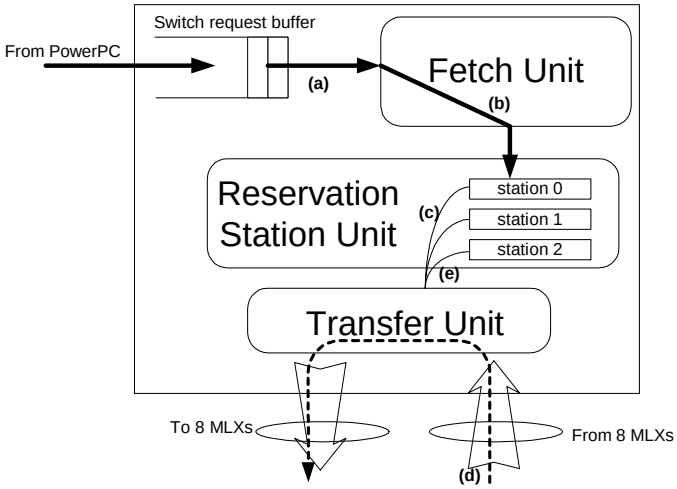


Figure 7 Switch controller

3.2. Message flow

Assume that two host processors to which NIs are connected will perform a remote DMA transfer in order to transfer a message from one host memory to another directly.

First, the message is prepared in the memory of sender's host processor. The PowerPC detects the message size, splits it into the size (2Kbytes) of the network buffer and transfers each chunk to a channel of network buffer one after another. After the first 32 byte of the chunk is written into the channel, MLX's protocol will begin to send packets to SB in burst. MLX's protocol decides the length of the burst at the beginning of the transfer. However, when the burst ends, if there are more packets in the channel, MLX's protocol puts a continuous word and continues to perform the next transfer without releasing the medium. MLX protocol module will try to send more packets in the network buffer until it confirms that the channel becomes empty. This activity can try to use the full bandwidth of the physical medium and achieves high throughput.

On SB, the packets from NI are stored into a channel of network buffer. At the end of the burst transfer, the MLX protocol module sends its acknowledge, if the checksum is correct, to the sender side. These communication steps will be repeated until the entire message will be transferred to SB.

During the message transfer from the sender's NI, message analyzer on SB will detect a coming message from NI after receiving the chunk of whole message. First 8 bytes of the message will be stored in the message header buffer, and PowerPC will detect the existence of the message header, read it from the buffer and analyze the destination of the message. After the decision of the destination of the message, PowerPC will write a switching request into the switch request buffer in the switch controller.

The switch controller's request fetch unit will read the switch request from the switch request buffer and tries to put it into a free reservation station. When the reservation station unit detects the storing of a request from the fetch unit, it will activate the request and make the transfer unit to process the request. The transfer unit will process the activated request to transfer as much packets as are stored in the source channel. If the source channel of the network buffer becomes empty or the destination channel of the network buffer becomes full, the transfer process will stop and inform the state of the transfer to be stored back into the reservation station. If there are more possible transfer requests in the reservation station unit, for example there is a request to transfer a message for which the source and the destination channels are different, the unit will try to resume the request and put it to the transfer unit.

Then, MLX of the destination channel on SB will begin to send the packets as well as the sender side. This is begun when the first 32 bytes of the message is stored into the network buffer. The MLX on the receiver's host will detect the receiving of the packets and setup a DMA transfer from the channel of the network buffer to the host memory. The DMA engine repeats sending packets of the message via PCI bus while the capacity of the receive channel is not empty.

On the whole process above, when the first 32byte of a message is transferred to each component, the transfer operation in each component will be activated one after another. Therefore, the transfer from the sender's host memory to a channel of the network buffer, the ones between NI's and SB's MLX protocol modules and the one in the switching controller will be processed in a pipelined manner. Thus, the latency of the switch will be overshadowed and high throughput will be achieved.

3.3. Communication software

To extract the potential performance of Maesto2 network hardware, we design and implement a dedicated message passing library software called MMP.

MMP is a message passing library that has performance tunings for Maestro2 hardware and a preparation for upper layers: 1) hardware control in the user level, 2) 0-copy communication and 3) asynchronous messaging mechanism.

1. Hardware control in the user level

   The conventional communication software such as TCP/IP never allows application software to touch the hardware without using socket interface[30]. In this case, the application software inevitably uses the interface that might includes the overhead such as implicit copy operations. In the case of MMP, the Maestro2 driver in the OS kernel allows application software that uses MMP library to

touch the hardware resource directly. The application software can skip all the communication layers and can touch the hardware resource directly, and thus will eliminate the implicit overheads inside of the conventional interface.

2. 0-copy communication

   MMP uses 0-copy mechanism to communicate the messages to avoid copy operation in the host machines. 0-copy mechanism used in the conventional commercial available software such as GM and PM. To implement 0-copy mechanism, MMP will use the driver to pindown memory pages in the host machine [24] and to send and receive messages from and to the memory pages directly.

3. Asynchronous messaging mechanism

   MMP library prepares an asynchronous transmission function pair (MMP_Send() function and MMP_Wait_send()) and an asynchronous reception function pair (MMP_Recv() and MMP_Wait_recv()). MMP_Send() function sends a message but never guarantees its completion. MMP_Wait_send() performs its synchronization. On the other hand, MMP_Recv() and MMP_Wait_recv() are the dual functions for receiving data. This mechanism allows the easily implementation of the middleware such as MPI[19], keeping the compatibility.

MMP is the lowest layer of the communication software hierarchy in the host machine but it is not compatible with the communication layer in the kernel because it is a user level library. In the case of MPI implementation, there should be two types of the lowest level: MMP and TCP. We can separate those with regard to the size of communication area. MPI over MMP can be suitable for a really small area communication in a cluster. MPI over TCP can be suitable for a communication among clusters.

4. Experimental results

For experimental evaluation of Maestro2 network, four categories of experiments will be done:

1. Basic performance evaluation
2. Continuous network burst evaluation
3. Out of order switching evaluation
4. Comparison with other Gbit-based networks

TCP/IP emulation is used for the third and fourth experiments. This emulation is implemented as a compatible firmware for the network device driver over Linux. The MTU size is 65000bytes. The bigger MTU size is the more throughput is achieved[35]. However, because the maximum size of IP packet is 64kbytes, we can not make the size bigger than the IP's limitation.

All the experiments in this section will be performed with the component shown in Table 1. The Gigabit Ethernet and Myrinet use the jumbo frame (i.e 9000byte) and 8Kbyte frame respectively.

The timebase register on PowerPC of NI is used for the measurement of the latency, in the first experiment. In the cases of MMP or MPI or TCP, the gettimeofday() function was used to obtain the results.

Table 1 Experimental environment

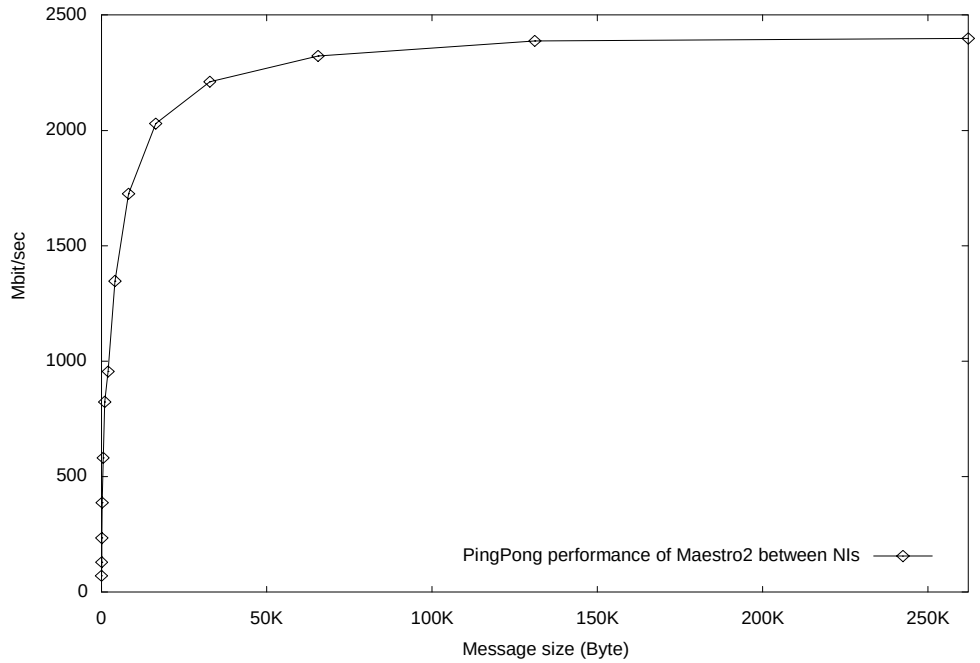
OS	RedHat Linux 2.4.18

Myrinet	Myrinet-2000 PCI64B (NIC) and M3F-SW16-8F(SW)
Gbit Ethernet	Intel PRO/1000MT Server adapter(82545) 64bit/66MHzPCI(NIC) DELL PowerConnect5224 (SW)
Host Computer	Supermicro 370DE6, PentiumIII 800MHz and 512MB 133MHz SDRAM

4.1. Basic performance

To measure the basic performance of network hardware itself, we used a pingpong communication pattern between two NIs via SB. In the pingpong communication pattern, after the sender's NI saves the value of timebase register, it sends a message from the SDRAM on the NI to the network buffer by DMA. When the receiver detects the message in its network buffer, it transfers from the network buffer to the SDRAM and then performs same way as the sender to return the message to the sender. Finally, the original sender stops and saves the value of timebase register and calculates the time difference between the beginning value and ending value of the timebase register. The result is divided by two (i.e. one way latency).

The basic performance evaluation is shown in Figure 8. The minimum latency is 3.6 microseconds. The maximum throughput is 2.4 Gbit/sec, witch corresponds to 75 % of the maximum spec allowed by physical medium.



Size(bytes)	32	64	128	256	512	1K	2K	4k
Latency(usec)	3	3	4	5	6	9	16	23

Figure 8 Basic performance of Maestro2

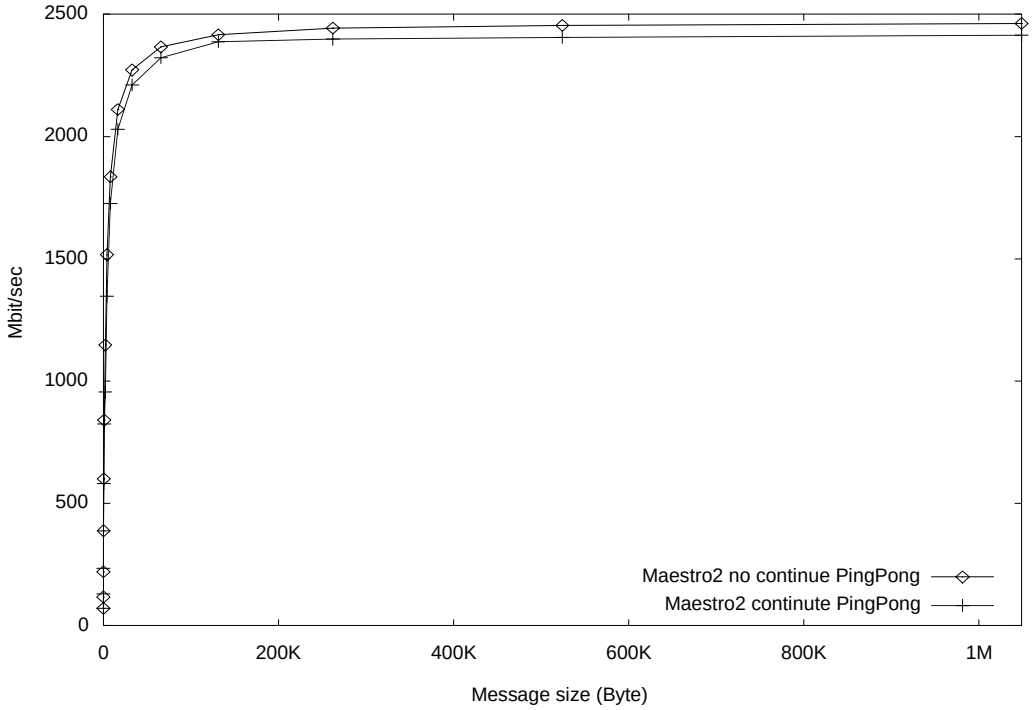
4.2. Continuous Network burst

To compare the performance achieved with and without continuous burst, the pingpong method was used

as in the basic performance measurement.

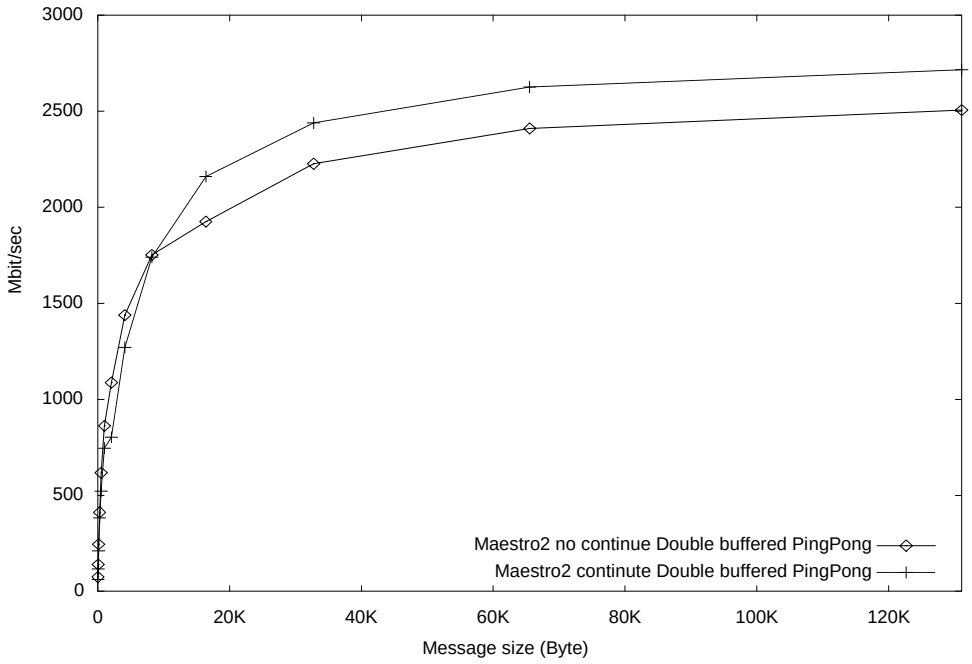
Figure 9 shows the throughput comparison of pingpong by using one channel of the network buffer. A frame transfer between MLXs completes after receiving the acknowledge. Before the acknowledge, if the sender's channel is full, the DMA transfer from the host machine's memory to the network buffer will stall until the acknowledge arrives to the sender. In the case of one channel utilization, stalls occur to fill a part of message into the channel by DMA transfer occur while the MLX waits to receive acknowledge from the other side of the connection. That is why the continuous one is slightly less than the non-continuous one.

Figure 10 shows the performance comparison by using double buffering pingpong. The double buffering pingpong is performed by splitting a message into the size of channel's buffer and writing the chunks of message into two channels alternately. 200Mbps higher performance of the continuous one than the non-continuous one has been confirmed from the graph. In this case, the data filling/extracting operation into/from a channel from/into the host memory and the long burst transfer at MLX are overshadowed. Thus, the throughput of continuous network burst achieves higher than the non-continuous one. This characteristic will be more effective when a parallel application program uses multiple connections of multiple channels such as collective communication.



Size(bytes)	32	64	128	256	512	1K	2K	4k
Continue's latency(usec)	3	3	4	5	6	9	16	23
Non continue's latency(usec)	3	4	4	5	6	9	13	21

Figure 9 Continuous vs. non-continuous burst by pingpong with single channel



Size(bytes)	32	64	128	256	512	1K	2K	4k
Continue's latency (usec)	3	4	4	5	7	10	18	25
No Contenue's latency(usec)	3	4	4	5	7	9	13	20

Figure 10 Continuous vs. non-continuous burst by double buffering pingpong

4.3. Out of order switching

For the evaluation between in-order switching and out-of-order switching, we used a pingping communication with MPI over MMP and TCP/IP with two nodes. On the pingping communication, two host machines send a message simultaneously with explicit synchronization. The first sender sends a synchronization message to the other, then the other side returns acknowledge of the synchronization message to the sender. After that, both sides begin to send messages to each other. When both sides finish sending the same size massages, they will have a synchronization as well as the one at the beginning. This communication pattern can emulates a message competing state on switch. We will measure the latency between those two synchronizations.

Figure 11 shows the performances of pingping communication for MMP with varying the number of reservation stations for the pending requests. When the size of message is less than 64Kbyte, the messages from both sides compete rarely. That causes the almost same performances between in-order and out-of-order methods. However, after 64kbyte, the messages compete often in SB. MPI sends at most four messages (data messages and acknowledges from both sides). That is why the performance saturates when the number of stations is more than four. On the other hand, in the result of MPI over TCP/IP depicted in Figure 12, four messages on switch compete while pingping communication at a transfer as well as the case on the experience with MMP. However, due to the unstable scheduling of TCP's communication timing, the timing in SB to receive and process messages is unpredictable. This makes its performance better with more then 4 stations even if the number of messages are same as the case of MMP.

We performed another experiment with two pairs of the same pingping communication pattern over MMP,

that is performed with four host machines. Figure 13 shows the maximum throughput when the numbers of stations are 1, 2, 4 and 8. In this experiment, the synchronization among four machines is performed by broadcasting a packet from SB to all the machines. This synchronizations are performed two times, at the beginning of pingping communication in each machine and the end of it. Each line in the graph shows the throughput calculated from the latency in one of the machines between two synchronizations. The performance improves up to 92% of the one of single station. When the number of stations is 4 or 8, the performance does not change because the maximal number of requests in this experiment to the switch controller is four at a same time.

Regarding to the size of hardware resource, Table 2 shows the number of SLICEs required by the switch logic on the implementation in a FPGA. The increase of logic from the in-order logic to the out-of-order one with 8 stations is 180,000 gates, that is, about 10% of all the logic with in-order logic in spite of the large performance improvement, achieved.

All in all, the out-of-order switching achieves higher throughput than the in-order switching. In addition, we confirmed the effectiveness of out of order switching becomes higher as the number of host machines, and computation for communication increases.

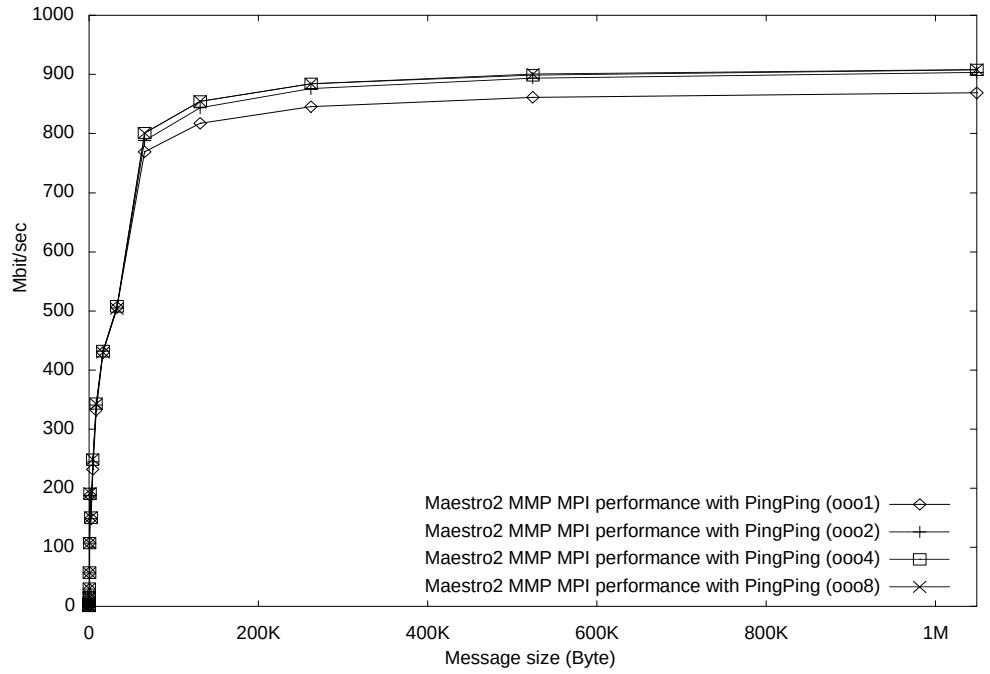


Figure 11 Out of order switching performance by MMP-MPI with two processors

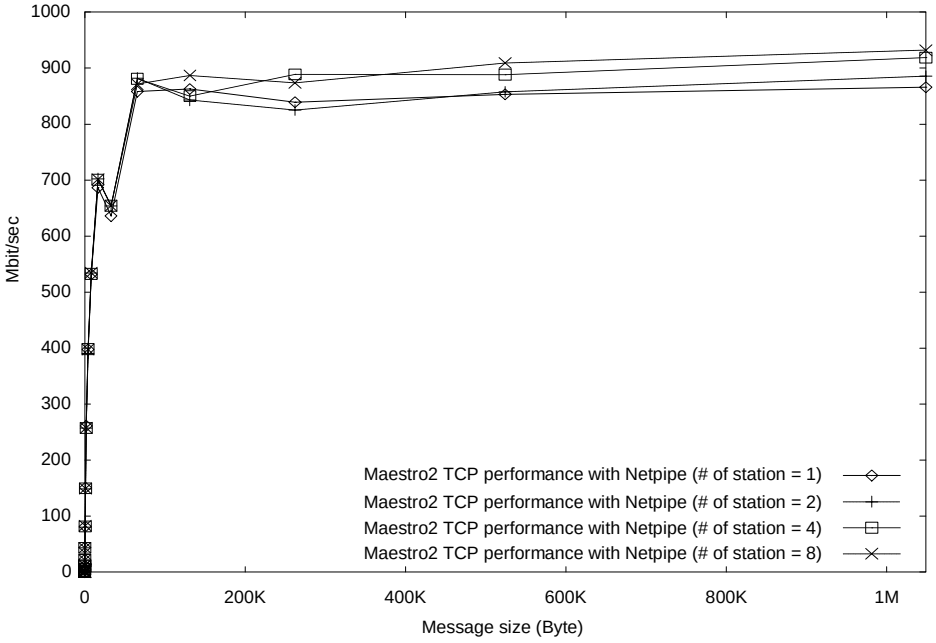


Figure 12 Out of order switching performance comparison by TCP/IP with two processors

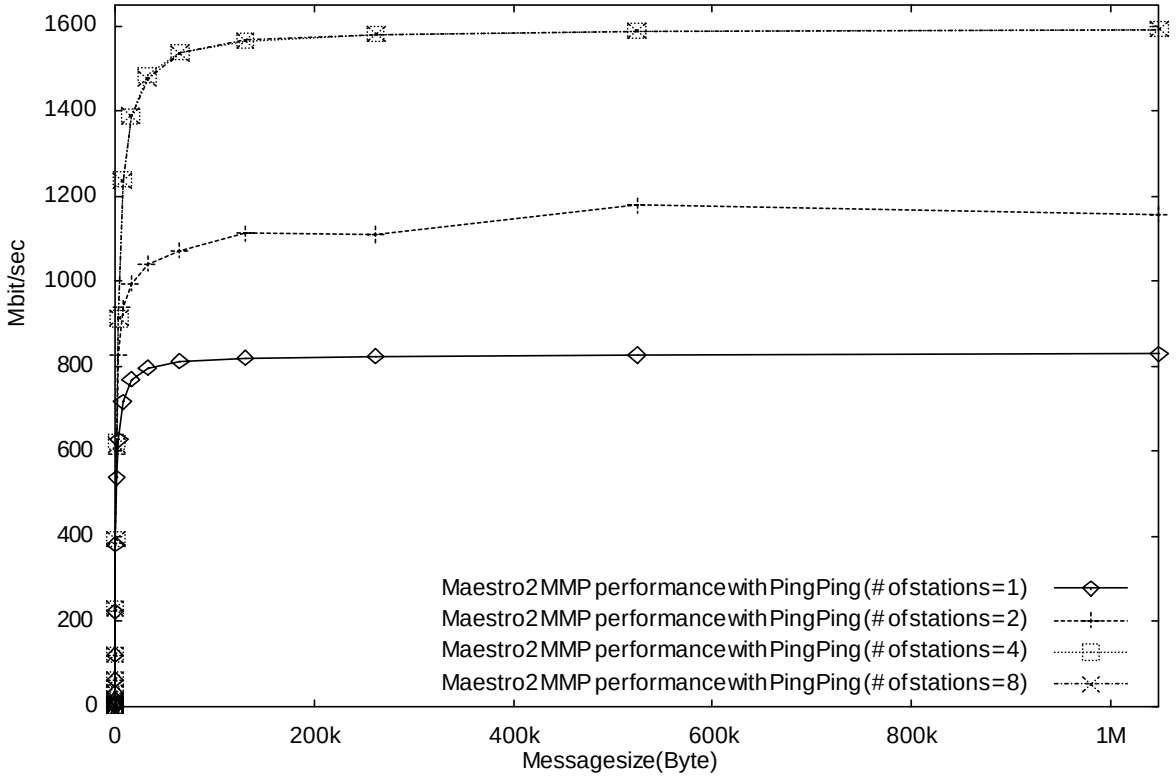


Figure 13 Out of order switching performance by MMP with four processors

Table 2 Logic gates needed for SW

# of stations	1	2	4	8
# of Slices	9135	9232	9790	10008
Times of 1 station	1.00	1.01	1.07	1.10
Estimated logic million gates	1.91	1.93	2.05	2.09

4.4. Comparison with other communication systems

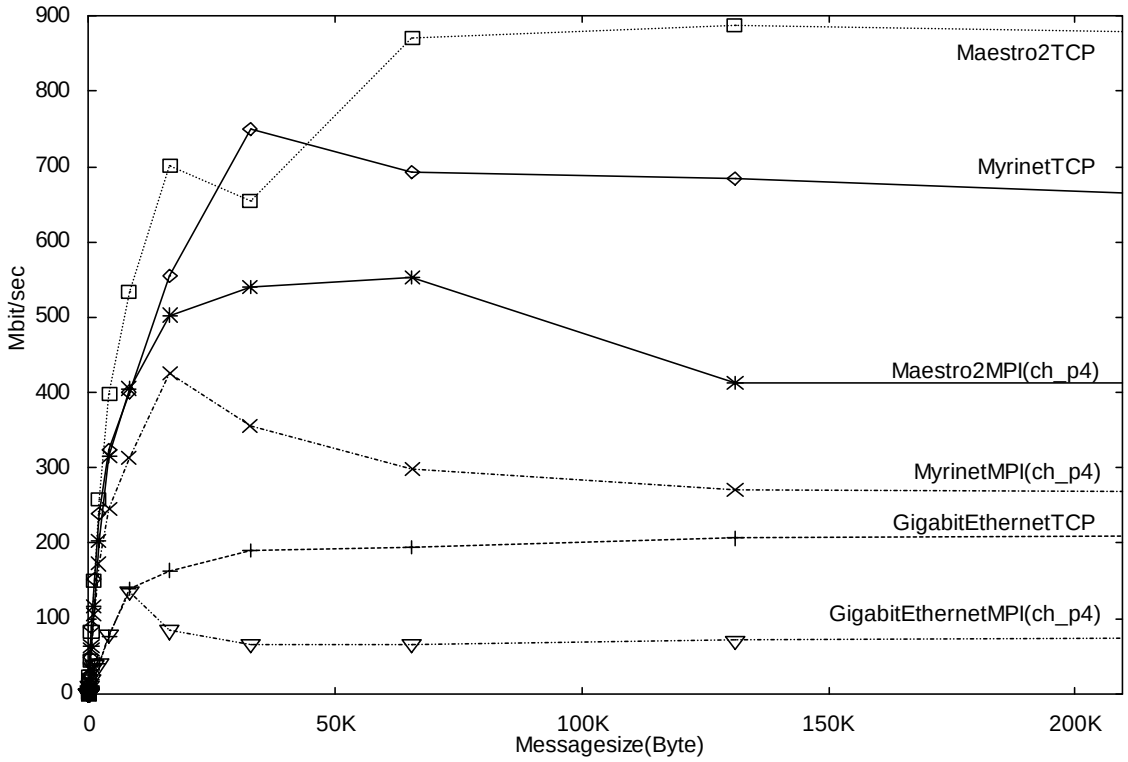
We compared Maestro2 performance with commercial available Gigabit Ethernet and Myrinet over TCP/IP, MPI over TCP/IP, GM and MPI over GM. GM is the dedicated message passing software that Myricom provides for Myrinet hardware.

Netpipe[29] was used to compare the performance of the three networks. Netpipe is a communication performance evaluation tool that counts the amount of data that has been migrated in a message at a specific time. In addition to the support for GM and MPI, Netpipe has a clear-layered architecture. We implemented MMP version of Netpipe without any change of Netpipe's communication algorithm.

Figure 14 shows the comparison among the three networks with MPI over TCP/IP and TCP/IP. *Ch_p4* is a low level library of MPICH[10] for socket-based communication with TCP/IP. In ch_p4 layer, copy operations are performed. This causes performance degradation, which is about 50-80% of TCP performance. This is the potential overhead of MPI over ch_p4. Moreover, it saturates after 64Kbyte message size due to the window size limitation of TCP.

The TCP's maximum throughput of Maestro2 is about 1.27 times and 1.3 times better than Gigabit Ethernet and Myrinet, respectively. The minimum latency of TCP over Maestro2 is about 1/8 than the one of the gigabit Ethernet. However, it is almost same as the one over Myrinet due to the overhead of TCP itself.

On the other hand, the MPI throughput with Netpipe of Maestro2 is about 4 times and 2 times higher than the one of Gigabit Ethernet and the one of Myrinet, respectively.

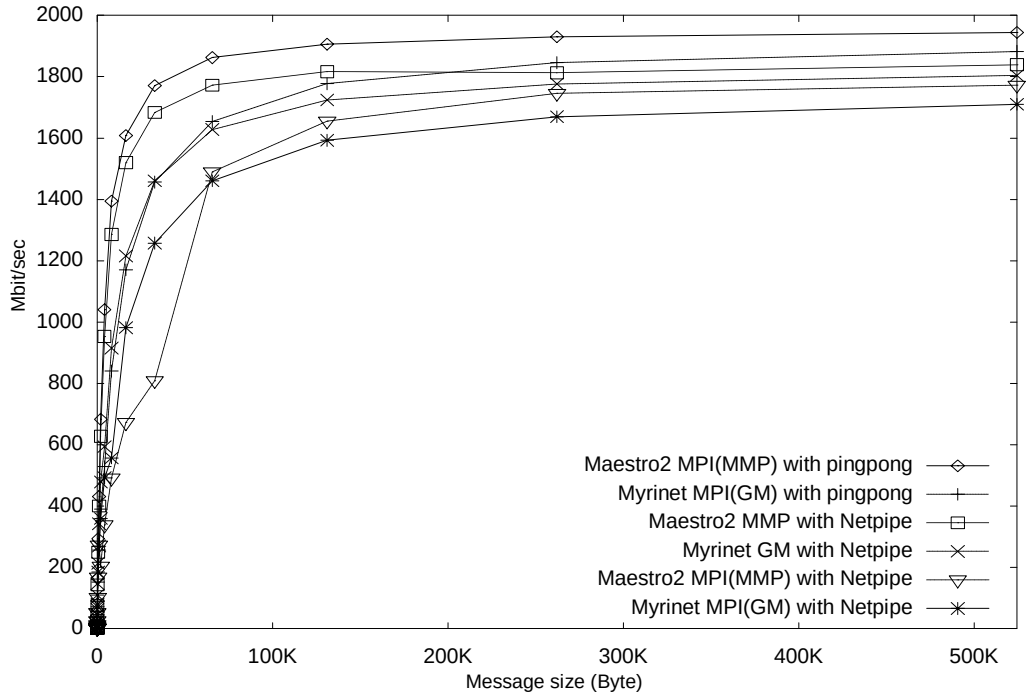


Size(bytes)	32	64	128	256	512	1K	2K	4k
Myrinet TCP latency(usec)	37	37	38	41	44	51	65	96
Gigabit Ethernet TCP latency(usec)	250	250	250	250	250	250	250	250
Maestro2 TCP latency(usec)	42	43	45	45	47	52	60	79
Myrinet MPI latency (usec)	53	55	56	57	66	75	91	128
Gigabit Ethernet MPI latency (usec)	250	250	250	250	250	250	250	271
Maestro2 MPI latency (usec)	57	57	58	61	63	68	77	99

Figure 14 Performance comparison by TCP among Gigabit Ethernet, Myrinet and Maestro2 by Netpipe

Using MMP and GM, we compared the performances with and without MPI layer by Netpipe as shown in Figure 15. In addition, pingpong program with MPI functions was also used for the measurement to see the performance without the overhead in the flow control of Netpipe. This pingpong communication pattern is the same we used in the section 4.1. The experimental results with this pingpong program are better than the performance with Netpipe due to elimination of the flow control.

Regarding the latency of small messages, all the results except the one without MPI over Netpipe show the performances of Maestro2 are better than Myrinet. In addition, we confirmed the result without MPI over Netpipe shows the big acceleration of utilization of physical medium with Maestro2, because the latency becomes almost half of the one of Myrinet when the message is 4Kbyte. According to the best results of MMP and GM with the pingpong program, we confirmed MMP performance was about 100Mbps better than the one of GM.



Size(bytes)	32	64	128	256	512	1K	2K	4k
Maestro2 MPI latency with pingpong (usec)	15	20	20	20	20	25	80	100
Myrinet MPI latency with pingpong (usec)	20	20	22	23	27	32	42	62
Maestro2 MMP latency with Netpipe(usec)	13	13	13	14	16	20	25	33
Myrinet GM latency with Netpipe(usec)	10	11	12	13	18	23	33	53
Maestro2 MPI latency with Netpipe(usec)	19	19	19	19	23	29	77	92
Myrinet MPI latency with Netpipe(usec)	11	11	14	18	21	29	44	64

Figure 15 Performance comparison by dedicated software between Maestro2 and Myrinet

All in all, the continuous network burst accelerates the usage of physical medium and the out of order switching mechanism processes messages concurrently to achieve peak performance of SB. Thus we can conclude that the techniques proposed on this paper are effective for improving cluster communication performance.

5. Conclusions

This paper is focused on the cluster network. Communication overheads using conventional network equipment were identified and new techniques to address those overheads were proposed.

To implement the proposed techniques, a network system Maestro was first developed. This Maestro network, was described and some experimental results were presented. Two additional issues that restricted the overall performance were identified.

As a follow-up work a new architecture was developed, called Maestro2. This paper described in detail Maestro2 design and implementation, with emphasis on two new improvement techniques to solve identified problems in the Maestro network, namely *Continuous network burst* and *Out of order switching*.

Experimental results were obtained to measure the performance, to evaluate the improvements achieved using the new techniques, and to compare Maestro2 with other gigabit-based networks. It is shown that Maestro2 technology is able to provide high performance, in terms of low latency and high throughput. Experimental results allow to state that Maestro2 is able to utilize up to 84% of the physical layer bandwidth. Using either TCP/IP, MPI or proprietary communication libraries (MMP for Maestro2 and GM for Myrinet) it was shown that Maestro2 performance is better than the main reference commercial network equipments.

The next steps in the development of the technology are to achieve improvement of best effort switching (using time divided and connection oriented strategies), to study scalability issues and to provide shared memory support.

References

[1] Keiichi Aoki, Shinichi Yamagiwa, Masaaki Ono, Koichi Wada and Luis Miguel Campos. *An Architecture of High performance cluster network: Maestro2*, In Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 2003

[2] Mark Baker, *Cluster computing white paper*, 2000

[3] N. J. Boden, D. Cohen, et al. *Myrinet: A Gigabit-per-Second Local Area Network*. IEEE Micro, pages 29-35, Feb 1995. http://www.myri.com

[4] Robert Breyer and Sean Riley. *Switched and Fast Ethernet, Second Edition*. Zi. Davis Press, 1996.

[5] Rajkumar Buyya. *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall, 1999.

[6] Rajkumar Buyya. *High Performance Cluster Computing: Programming and Applications*. Prentice Hall, 1999.

[7] Rohit Chandra, Leo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald and Ramesh Menon, *Parallel Programming in OpenMP*, Morgan Kaufmann, 2000

[8] von Eicken, A. Basu, and W. Vogels. *U-Net: A user level network interface for parallel and distributed computing*. In Fifteenth ACM Symposium on Operating Systems Principles, pp. 40-53, 1995.

[9] John L. Hennessy and David Patterson, *Computer Architecture: A Quantitative Approach* Section 3.3 , Morgan Kaufmann Publishers, 1990

[10] William Gropp, E. Lusk, N. Doss and A. Skjellum, *A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard*, MPI Developers Conference, 1995. http://www-unix.mcs.anl.gov/mpi/mpich/

[11] IEEE Standard Department. *IEEE Standard for Low-Voltage Differential Signals (LVDS) for Scalable Coherent Interface (SCI)*, 1996.

[12] IEEE Standard Department, *IEEE standard for a High Performance Serial Bus*, 1994. http://www.1394ta.org.

[13] IEEE Computer Society, Task Force on Cluster Computing, http://www.clustercomputing.org/

[14] P. Keleher, S. Dwarkadas, A.L. Cox, and W. Zwaenepoel, *TreadMarks: Distributed Shared Memory on*

Standard Workstations and Operating Systems, Proceedings of the Winter 94 Usenix Conference, pp. 115-131, January 1994.

[15] H. T. Kung, T. Blackwell, and A. Chapman. *Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation and Statistical Multiplexing*. In ACM SIGCOMM, August 1994.

[16] Daniel E. Lenoski and Wolf-Dietrich Weber, *Scalable Shared-Memory Multiprocessing*, Morgan Kaufmann, 1995

[17] Scott Pakin, Vijay Karamcheti, and Andraw A. Chien. *Fast messages (FM): E.cient, portable communication for workstation clusters and massivelyparallel processors*. IEEE Concurrency, Vol. Vol.5, No. No.2, pp. 60-73, 1997.

[18] Loic Prylli and Bernard Tourancheau. BIP: a new protocol designed for high performance networking on myrinet. In Workshop PC-NOW, IPPS/SPDP98, Orlando, USA, 1998. Elsevier Science Publishers.

[19] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, 1995 http://www-unix.mcs.anl.gov/mpi/mpich/

[20] Motorola. *MPC603e & EC603e Microprocessor User's Manual*. 1997.

[21] Myricom, Inc. *The GM Message Passing System*. 1999.

[22] PCI Special Interest Group. *PCI Local Bus Specification. Rev. 2.1*. 1995.

[23] Daniel Ridge, Donald Becker, Phillip Merkey, Thomas Sterling Becker, Phillip Merkey, *Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs*, Proceedings, IEEE Aerospace, 1997

[24] Alessandro Rubini and Jonathan Corbet, *Linux Device Drivers, 2nd Edition*, O'Reilly & Associates, June 2001

[25] Stephen Saunders, Gigabit Ethernet (McGraw-Hill Computer Communications Series), McGraw-Hill Professional, 1998

[26] Issac D. Scherson, *Proceedings: The New Frontiers: A Workshop on Futurre Directions of Massively Parallel Processing*, IEEE Computer Society, 1992.

[27] T. Shanley, *Infiniband Network Architecture*, Addison-Wesley, November 2002.

[28] Sinnen, L. Sousa, *List Scheduling: Extension for Contention Awareness and Evaluation of Node Priorities for Heterogeneous Cluster Architectures*, Parallel Computing, January 2004.

[29] Quinn O. Snell, Armin R. Mikler, and John L. Gustafson,*NetPIPE: A Network Protocol Independent Performance Evaluator*, IASTED Conference Paper. http://www.scl.ameslab.gov/netpipe/

[30] W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, Richard W. Stevens, *Unix Network Programming, Vol. 1: The Sockets Networking API*, Addison-Wesley, October 2003.

[31] Hiroshi Tezuka, Atsushi Hori, Yutaka Ishikawa, and Mitsuhisa Sato, *PM: An Operating System Coordinated High Performance Communication Library*, In Peter Sloot and Bob Hertzberger, editor, High-Performance Computing and Networking, Vol. 1225 of Lecture Notes in Computer Science, pp. 708-717, Springer-Verlag, April 1997.

[32] Xilinx Inc, *Virtex II - platform FPGA Data Sheet* 2002. http://www.xilinx.com

[33] Shinichi Yamagiwa, Munehiro Fukuda, Koichi Wada, *Design and Performance of Maestro Cluster Network*, Proc. of the IEEE Int'l Conference on Cluster Computing - Cluster2000

[34] Shinichi Yamagiwa, Luis Miguel Campos, Keiichi Aoki, Masaaki Ono, Tetsuya Sakurai and Koichi

Wada. *Maestro2: A new challenge for high performance cluster network*. In proceeding of the 6th World Multiconference on Systemics, Cybernetics and Informatics, July 2002

[35] Shinichi Yamagiwa, Kevin Ferreira, Luis Miguel Campos, Keiichi Aoki, Masaaki Ono, Koichi Wada, Munehiro Fukuda, Loenel Sousa. *On the Performance of Maestro2 High Performance Network Equipment, Using New Improvement Techniques*. In 23rd IEEE International Performance Computing and Communications Conference(IPCCC 2004), 2004.

[36] Koichi Wada, Shinichi Yamagiwa, Munehiro Fukuda, "High Performance Network of PC Cluster Maestro", Cluster Computing, Volume 5, Issue 1, January 2002, Pages 33 - 42