

Implementation and Performance Study of a Hardware-VIA-based Network Adapter on Gigabit Ethernet

Sejin Park^a, Sang-Hwa Chung^{a,*}, Ben Lee^b

^a*Department of Computer Engineering, Pusan National University, Pusan 609-735, Korea*

^b*School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA*

Abstract

This paper presents the implementation and performance of a hardware-VIA-based network adapter on Gigabit Ethernet. VIA is a user-level communication interface for high performance PC clustering. The network adapter is a 64-bit/66MHz PCI plug-in card containing an FPGA for the VIA Protocol Engine and a Gigabit Ethernet chip to construct a high performance system area network. The network adapter performs virtual-to-physical address translation, doorbell, RDMA write, and send/receive completion operations in hardware without kernel intervention. In particular, the Address Translation Table (ATT) is stored on the local memory of the network adapter, and the VIA Protocol Engine efficiently controls the address translation process by directly accessing the ATT. In addition, Address Prefetch Buffer is used to reduce the time of address translation process in the receiver. As a result, the communication overhead during send/receive transactions is greatly reduced. Our experimental results show a minimum latency of 8.2 μ s, and a maximum bandwidth of 112.1 MB/s. In terms of minimum latency, the hardware-VIA-based network adapter performs 2.8 times and 3.3 times faster than M-VIA, which is a software implementation of VIA, and TCP/IP, respectively, over Gigabit Ethernet. In addition, the maximum bandwidth of the hardware-VIA-based network adapter is 24% and 55% higher than M-VIA and TCP/IP, respectively. These results show that the performance of HVIA-GE is far better than that of ServerNet II, which is a hardware version of VIA developed by Tandem/Compaq.

Key words: VIA, Gigabit Ethernet, PC Clustering, User-Level Communication Interface, RDMA

* Corresponding author. Tel.: +82-51-510-2434; fax: +82-51-517-2431

Email addresses: sejnpark@pusan.ac.kr (Sejin Park), shchung@pusan.ac.kr

1 Introduction

As cluster computing becomes more popular due to the increase in network speed and the enhanced performance of computing nodes, a significant effort has been made to reduce the communication overhead between cluster nodes to maximize the overall performance. In particular, there have been much research efforts in user-level communication to minimize the kernel intervention, such as context switching and data copy between protocol layers. Examples include Active Messages [1], Fast Messages [2], VMMC [25], U-Net [3], and VIA [4]. Among them, Virtual Interface Architecture (VIA) was proposed to standardize different features of existing user-level protocols for System Area Networks (SANs). VIA can be implemented in either software or hardware. Successful implementations include M-VIA [5], Berkeley VIA [6], Firm VIA [16][17], ServerNet II [7] and cLAN [8]. Recently, InfiniBand Architecture (IBA) [18] which evolved from VIA, has been presented and a number of related works have been published [28][29]. In contrast to VIA, IBA has specific definitions for interconnections such as links, routers and switches. IBA products such as host and target channel adapters and InfiniBand switches are available from Mellanox Technologies [28].

Since VIA is a very flexible specification to implement, the prior work on VIA have different design approaches and various network platforms. In this paper, Gigabit Ethernet is adopted as an underlying network to construct a VIA-based PC cluster. Since Gigabit Ethernet is a standard high-speed network for LANs, it has an advantage in terms of cost when compared with proprietary high performance networks, such as Myrinet [9] and InfiniBand. Moreover, when VIA is adopted as a user-level interface on Gigabit Ethernet based clusters, most of the low-level bandwidth can be redeemed at the application level by removing the time consuming TCP/IP protocol. Recently, there have been a number of efforts to implement software versions of VIA based on Gigabit Ethernet using either M-VIA or Berkeley VIA [11][12][13][15].

On the other hand, Tandem/Compaq developed a hardware version of VIA, called ServerNet II [7], using Gigabit Ethernet as a physical network. ServerNet II uses its own switches that support wormhole routing with 512-byte packets to connect cluster of nodes. ServerNet II has a minimum latency of 12 μ s for 8-byte data and a bandwidth of 92 MB/s for 64 KB data using RDMA (Remote Direct Memory Access) writes on a single Virtual Interface channel. Although, the specific details of the implementation were not reported, the address translation table was not implemented in hardware because there is no memory on the card. cLAN is also a hardware implementation of VIA [8], which has a minimum latency of 7 μ s and a maximum bandwidth of 110 MB/s.

(Sang-Hwa Chung), benl@eecs.orst.edu (Ben Lee).

Although cLAN shows better performance than ServerNet II, it is based on an expensive proprietary network, similar to Myrinet and InfiniBand.

This paper presents the design and implementation of HVIA-GE, which is a *Hardware* implementation of *VIA* based on *Gigabit Ethernet*. HVIA-GE is a PCI plug-in card based on 64-bit/66MHz PCI bus. An FPGA was used to implement the VIA Protocol Engine and a Gigabit Ethernet chip was used to connect the VIA Protocol Engine to Gigabit Ethernet. HVIA-GE performs virtual-to-physical address translations, send/receive operations including RDMA, and completion notifications fully in hardware without any intervention from the kernel. In particular, the Address Translation Table (ATT) is stored in the local memory of the HVIA-GE card, and the VIA Protocol Engine efficiently performs the virtual-to-physical address translations. In addition, Address Prefetch Buffer is used to reduce the time of address translation process in the receiver. The PCI logic was directly implemented on the FPGA rather than using a commercial chip to minimize the latency of DMA initialization. The HVIA-GE cards can be connected to Gigabit Ethernet switches developed for LANs to form a cluster; therefore, a high performance but low cost cluster system can be easily constructed.

This paper is organized as follows. Section 2 briefly overviews VIA, and Section 3 describes the implementation details of HVIA-GE. Section 4 discusses the experimental results of HVIA-GE. Finally, section 5 provides a brief conclusion and a description of future work.

2 VIA Overview

VIA supports low-latency, high-bandwidth communications on SANs. Figure 1 shows the organization of the VI Architecture. VIA uses the Virtual Interfaces (VIs) to reduce the communication overhead. A VI for each node functions as a communication endpoint, and VIs generated between two nodes establish a virtual communication channel. VI Kernel Agent provides the necessary kernel services, which include connection management and memory registration, to establish VI connections between nodes. VI User Agent provides routines for data transfer, connection management, queue management, memory registration, and error handling.

Each VI contains a Work Queue (WQ), which consists of a Send Queue and a Receive Queue. A send/receive transaction is initiated by posting a VI descriptor on the WQ, and the Network Interface Card (NIC) is notified of the send/receive transaction using a doorbell mechanism. Each VI descriptor contains all the information the NIC needs to process the corresponding request, including control information and pointers to data buffers. Then, the NIC per-

forms the actual data transfer using DMA without any interference from the kernel. The send/receive transaction is completed when the VI descriptor's done bit is set and the Completion Queue (CQ) is updated by setting the corresponding VI descriptor handle.

The VIA standard provides two types of communication methods: the traditional send/receive and the RDMA read/write. In case of the RDMA, the initiator node of the RDMA operation specifies the virtual address of the local and remote memory to be read or written. Then, data on the remote node can be read or written without intervention from the remote processor. The memory region used for data transfer must be pinned down when it is registered to prevent it from being swapped out during send/receive or RDMA. Thus, the NIC can read/write data to the memory region safely and directly.

The VIA supports three levels of communication reliability: unreliable delivery, reliable delivery, and reliable reception. Unreliable delivery only guarantees that a send or an RDMA write is delivered at most once to the receiving side. A reliable delivery guarantees that all data will arrive at its destination exactly once, intact, and in order. In reliable reception, a data transfer is successful only when the data have been delivered to the target memory location. This reliability level is an attribute of a VI and only VIs with the same reliability level can be connected. Support for reliable delivery and reliable reception is optional as defined by the VIA specification 1.0 [4]. Thus, our implementation is based on unreliable delivery.

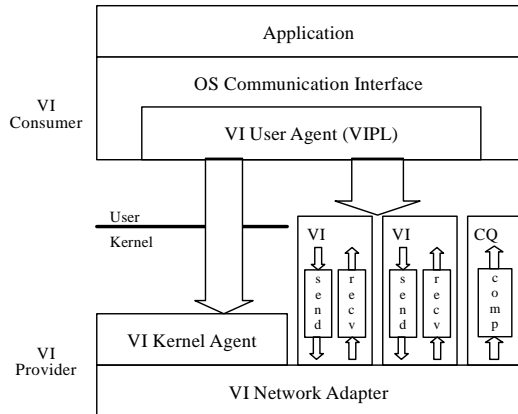


Fig. 1. VI architecture

3 Implementation of HVIA-GE

There are many possible approaches for implementing the mechanisms of WQ, CQ, doorbell, and address translation [14][16]. While it is possible to implement them in either software or hardware, the choice depends on both the

performance needs and cost. Even if a hardware implementation is chosen, there are many ways to implement each component.

Our implementation of the major components of VIA is based on the following motivations. First, it is not possible to store all the VI descriptors of WQ on the local memory of the HVIA-GE card because the number of VI descriptors can increase greatly depending on the characteristics of each application. Therefore, WQ is stored in the host memory but the VI descriptors that are currently being processed are copied and stored in HVIA-GE until the corresponding send/receive transactions are completed. Second, the send/receive completion notification mechanism is implemented only using the "done bit" in the status field of the VI descriptor. If CQ is to be used for the completion notification mechanism, it would be necessary to maintain CQ on the host memory together with WQ, which will result in an extra DMA operation to update the corresponding VI descriptor in CQ. Third, the doorbell mechanism that notifies the start of a send/receive transaction is implemented using registers in the HVIA-GE card. Finally, since every send/receive operation requires a virtual-to-physical address translation, ATT is stored on the local memory implemented on the HVIA-GE card. This allows the VIA Protocol Engine to efficiently control the address translation process based on the ATT.

Figure 2 shows the block diagram of the HVIA-GE card, which is a network adapter based on 64-bit/66MHz PCI bus. The PCI interface logic, the VIA Protocol Engine, the SDRAM controller, and the Gigabit Ethernet controller are all implemented using an FPGA running at 66 MHz. The PCI interface logic is implemented directly on the FPGA, rather than using a commercial chip, such as PLX Technology's PCI 9656 [22], to minimize the latency of the DMA initialization. If a commercial PCI chip is adopted, a local bus is necessary to connect the PCI chip with the VIA Protocol Engine. Thus, the latency will increase since the information needed for the DMA initialization has to move through the local bus. Intel's Gigabit Ethernet Controller with integrated PHY (82544EI) is used to connect the card to Gigabit Ethernet [21]. On the software side, the Virtual Interface Provider Library (VIPL) and the device driver were developed based on Linux kernel 2.4. The following subsections provide the specifics of the HVIA-GE implementation.

3.1 VIA Protocol Engine and Gigabit Ethernet Controller

Figure 3 presents the VIA Protocol Engine and the Gigabit Ethernet Controller (GEC), which are the core modules of HVIA-GE. The VIA Protocol Engine consists of Send/Receive FIFOs, ATT Manager, Protocol Manager, RDMA Engine, Doorbells, and local memory controller. It processes VIPL functions delivered to HVIA-GE through the PCI bus. In the case of *VipReg-*

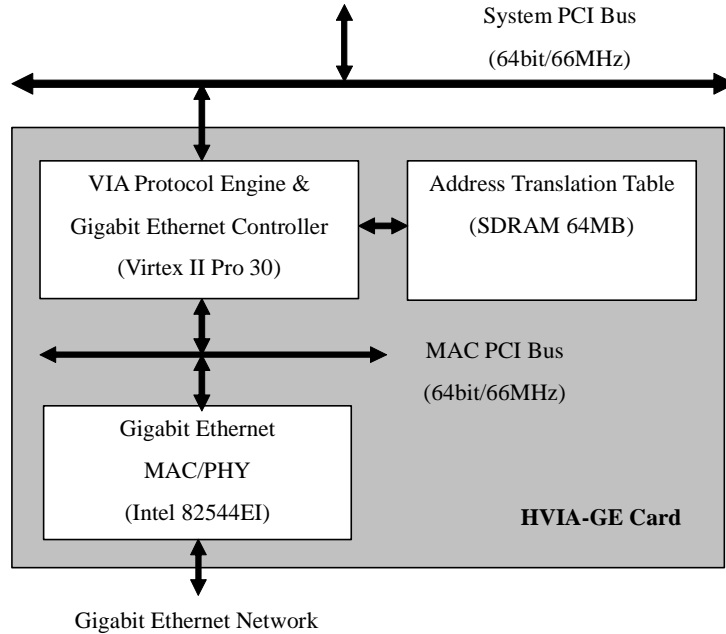


Fig. 2. HVIA-GE Card block diagram

isterMem, which is the VIPL function used to perform memory registration of a user buffer, the user buffer's virtual address, physical address, and size are sent to HVIA-GE as function parameters. The ATT manager receives information regarding the user buffer (i.e., virtual and physical addresses) and stores them on ATT.

When a send/receive request is posted to a send/receive queue, HVIA-GE is notified through the doorbell mechanism, and obtains the corresponding VI descriptor via DMA. Then, the VIA Protocol Manager reads the physical address of the user data through the ATT Manager. If the current transaction is a send, it initiates a DMA read operation for the user data in the host memory and transfers the data to the Tx buffer in the GEC via the Send FIFO. A send/receive transaction can also be implemented using RDMA, which enables a local CPU to read/write directly from/to the memory in a remote node without intervention of the remote CPU. An RDMA can be implemented as either RDMA read or RDMA write. If RDMA read is used, the local CPU must first send the request and then wait for the requested data to arrive from the remote node. Therefore, RDMA Engine in HVIA-GE is based on RDMA write, which is more advantageous in terms of latency.

Since HVIA-GE directly drives the Medium Access Control (MAC), GEC basically functions as a device driver for the MAC. GEC processes the initialization, transmit/receive, MAC management routines, and interfaces with the MAC using PCI. Although accessing the MAC directly complicates the design of the GEC and its internal buffers, the elimination of device driver

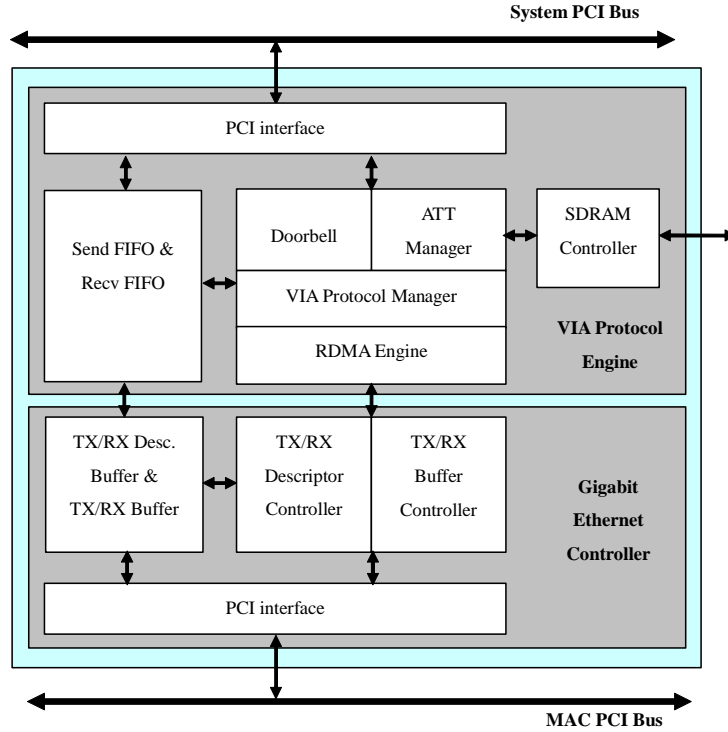


Fig. 3. VIA Protocol Engine & GEC block diagram

access reduces the initialization overhead and the actual transmission time.

The operations of GEC are as follows: When a send transaction is executed, Tx Descriptor Controller receives the size of the data to be transmitted and the address of the remote node from the VIA Protocol Engine, and produces a Tx descriptor. Meantime, Tx Buffer Controller adds the header information to the data received from the Send FIFO, stores the packet on the Tx Buffer, and informs MAC of the start of a transmission. Then, MAC reads the packet from the Tx Buffer and transfers it to the PHY. On the other hand, when a new packet arrives from a remote node, MAC refers to the corresponding Rx descriptor stored previously, transfers the data packet to the GEC's Rx Buffer, and updates the Rx descriptor. Afterwards, Rx Descriptor Controller determines that the packet has been received by polling the Rx Descriptor, and forwards the data to the Receive FIFO. Finally, the data is DMAed to the host memory.

3.2 Address Translation

During a VIA send operation, the user data is transmitted directly from the sender's user buffer to the receiver's user buffer without producing a copy in the kernel memory. To support this zero copy mechanism, the following

features must be implemented. First, once a user buffer is allocated for a send/receive operation, the virtual and physical addresses of the user buffer must be obtained and sent to ATT on the HVIA-GE card using PIO. Second, the user buffer must be pinned down when it is registered to prevent it from being swapped out during send/receive operations. In our implementation, one of the Linux kernel's features, KIOBUF (kernel I/O buffer) [20], is used to pinned down the user buffer. KIOBUF is an efficient mechanism to implement reliable memory locking [19]. KIOBUF makes it possible to map the virtual address of the allocated user buffer to the kernel area by calling the `map_user_kiobuf` function. Then, the user buffer is pin down by using the `lock_kiovec` function of KIOBUF. The virtual address and the corresponding physical address of the user buffer obtained during the pin down process are saved on ATT of the HVIA-GE card.

Figure 4 shows the ATT structure implemented in HVIA-GE. ATT is divided into ATT Level 1 and ATT Level 2. Each 24-byte entry of ATT Level 1 corresponds to one of the allocated user buffers, which includes the number of physical pages of the user buffer, the virtual address and the size of the first page, and ATT Level 2 pointer. In addition, ATT Level 1 stores the memory attributes needed for memory protection checking and RDMA read/write. ATT Level 2 stores the physical addresses (4-byte each) of all the allocated pages for the corresponding user buffer.

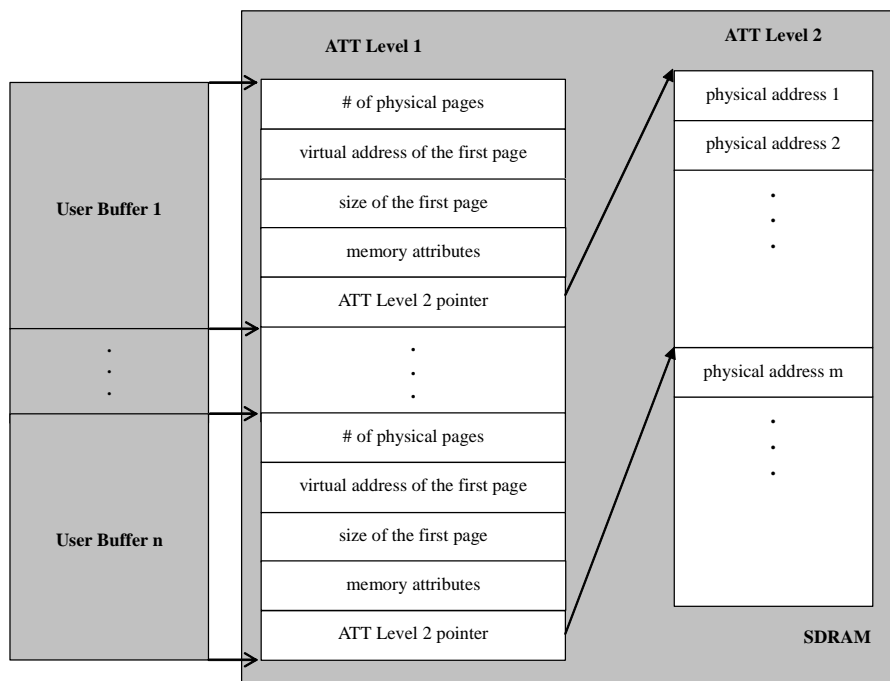


Fig. 4. ATT structure

Since ATT is implemented on the HVIA-GE card, it is important to acquire enough space for the table and provide an efficient access mechanism. In our implementation, a 64 MB SDRAM is used to store the ATT. If ATT supports

1024 VIs and each VI uses one user buffer, then SDRAM can support up to 60 MB of user buffer for each VI. If only one page (4 KB) is allocated for each user buffer, SDRAM can support more than 2.4 million user buffers. Therefore, the capacity of ATT should be sufficient to support most practical applications.

The access mechanism to ATT operates as follows. The kernel agent assigns a unique memory handle number to each user buffer in a linear fashion when it is allocated. An ATT Level 1 entry is also assigned in the same fashion by consulting the memory handle of the user buffer. Thus, the address of the entry can be calculated by multiplying the memory handle number by the entry size. The current ATT Level 2 pointer is calculated by adding the previous ATT Level 2 pointer to the number of the pages of the previous entry.

After a send/receive request is posted, HVIA-GE obtains the corresponding VI descriptor from WQ via DMA. The VI descriptor includes the corresponding memory handle and the virtual address of the user data. Then, the VIA Protocol Engine reads the corresponding ATT Level 1 entry using the memory handle. This requires only one SDRAM access to read the entire 24-byte entry in burst mode. The target address for the physical address at ATT Level 2 is determined by adding the ATT Level 2 pointer in the entry to the offset of the given virtual address. When a user buffer is allocated on the host memory, the start address of the user buffer can be at any place in the first page, which is indicated by the size of the first page in the entry. Thus, the size of the first page in the entry must be considered to properly determine the correct target address of ATT Level 2. Finally, the physical address of the user data is obtained by adding the physical address found at ATT Level 2 to the offset of the given virtual address. Therefore, the virtual-to-physical address translation can be processed using two SDRAM accesses.

Unlike the approach described here, it is also possible to implement ATT on the host memory. In this case, the VIA Protocol Engine has to access ATT via DMA reads. Although this method has an advantage in terms of the hardware cost, it takes about three times longer to access ATT on the host memory than SDRAM on the HVIA-GE card. Since ATT needs to be accessed for each send/receive operation, this overhead is significant, particularly when an application involves frequent communications between nodes in a cluster. More importantly, since ATT is copied from the host memory to SDRAM during memory registration in our implementation, this cost is not incurred during the actual send/receive operations.

Our design focused on minimizing the address translation overhead by removing the kernel intervention and the host memory accesses. However, it is possible that user buffers are deregistered as an application runs. This may cause fragmentation at ATT Level 2, and thus, some type of memory compaction scheme is needed.

3.3 *Send/Receive vs. RDMA Write Operations*

It is well known that the performance of an RDMA write operation is superior to that of a send/receive operation [30]. The reason is that for RDMA write operation the sender adds the virtual address of a remote node's user buffer to the address segments of the VI descriptor. Therefore, when a data packet arrives, the receiver performs address translation using only the virtual address information provided by the sender. However, in the case of a send/receive operation, the receiver needs to first call the `VipPostRecv` function. That is, the receiver has to create and post a VI descriptor, process the address translation, write data to the user buffer, and process completion for the VI descriptor. Therefore, the send/receive operation has extra overhead at the receiver causing the difference in performance.

However, the send/receive operation can be improved by modifying the address translation process in the receiver using the VI descriptor information generated by `VipPostRecv`. Since a receive operation calls `VipPostRecv`, the VIA Protocol Engine can determine the data size and the virtual address for the user buffer in advance. Then, the VIA Protocol Engine calculates the physical address of the user buffer using the virtual address and stores the physical address in an *Address Prefetch Buffer*, which can be accessed directly. Therefore, the time of address translation is reduced at the receiver.

The Address Prefetch Buffer contains entries to store two physical addresses. This is because the received data is in the form of an Ethernet packet and one packet can be stored in two pages. When the VIA Protocol Engine completes the processing of the current packet, it generates the two physical addresses of the next data packet in advance and stores them in the Address Prefetch Buffer. This is completed before the next data packet arrives. When the data arrives, the VIA Protocol Engine writes the data to the user's buffer using the addresses in the Address Prefetch Buffer instead of ATT.

RDMA write operations cannot take advantage of the Address Prefetch Buffer because the receiver does not know in advance the virtual address of the user buffer. Thus, it only improves the speed of address translation at the receiver for send/receive. Note that the sender's data transmission rate does not change because the Address Prefetch Buffer is only applied to the receiver, and the arrival times of data packets at the receiver are the same as before. Therefore, the performance advantage of the Address Prefetch Buffer is seen only for the last data packet, and the performance improvement is uniform regardless of data size. However, its effect can be significant for small message transfers, compared with no Address Prefetch Buffer.

3.4 Doorbell and Completion Mechanism

A doorbell is a notification mechanism that allows user processes to inform the network adapter when new send/receive transactions are posted. In existing software implementations, such as M-VIA, a doorbell is set by marking the doorbell region allocated in the user address space of the host memory, or by marking a flag in the corresponding VI descriptor. For the hardware implementation, a considerable amount of time will be required if the VIA Protocol Engine has to poll the doorbell region in the host memory, especially when many send/receive transactions are posted. In our design, the doorbell region is implemented using registers in the HVIA-GE card and user processes are allowed to access the region. Therefore, a user process can set the doorbell on the card as soon as a VI descriptor is posted to WQ. Consequently, the VIA Protocol Engine can quickly determine when a new send/receive request is posted by polling the doorbell, thereby reducing the latency.

There are two possible ways to implement the completion mechanism of a send/receive operation. First, it can be implemented by setting the done bit in the VI descriptor's status field and updating the corresponding VI descriptor in the CQ. Second, it can be implemented using only the done bit. If the second method is used, CQ is not involved in the completion mechanism and `VipCQDone/Wait` is not necessary. Moreover, the first method requires an additional DMA to update the VI descriptor in the CQ, and thus the second method was adopted in HVIA-GE. The user can then recognize completions by using polling functions (`VipSendDone/VipRecvDone`) or blocking functions (`VipSendWait/VipRecvWait`). Polling can recognize completions more rapidly because there is no interrupt processing overhead. However, CPU utilization degrades as data size grows. Therefore, blocking is more effective for large message transfers.

4 Experimental Results

The performance of the HVIA-GE card was evaluated using two 2 GHz Opteron 246 PCs with 64-bit/66MHz PCI bus [23]. The PCs were running Fedora core 1 for x86_64 with Linux kernel 2.4.22. In addition, for comparison purposes, the performances of TCP/IP and M-VIA were measured using an Intel PRO/1000 MT Gigabit Ethernet card [24]. This card includes Intel's Gigabit Ethernet Controller with integrated PHY (82544EI), which is the same chip with HVIA-GE's. Latency and bandwidth of the HVIA-GE were measured using a ping-pong program developed using VIPL. The performance of the M-VIA was measured using the `vnettest` program included, by the distributors, with the M-VIA. The performance of TCP/IP was measured by modifying

the `vnettest` program using the socket library.

4.1 Performance comparison of HVIA-GE, M-VIA, and TCP/IP

4.1.1 Latency and bandwidth

Figures 5 and 6 show the latency and the bandwidth results of HVIA-GE, M-VIA, and TCP/IP with an Ethernet MTU size of 1,514 bytes. The latency reported is one-half the round-trip time and the bandwidth is the total message size divided by the latency. The latency and bandwidth of the HVIA-GE were measured using send/receive on a single VI channel.

The minimum latency results of HVIA-GE, M-VIA, and TCP/IP are $8.2 \mu\text{s}$, $21.3 \mu\text{s}$, and $27 \mu\text{s}$, respectively. Therefore, the minimum latency of HVIA-GE is 2.8 and 3.3 times lower than M-VIA and TCP/IP, respectively. The maximum bandwidth results for 1 MB of user data are 112.1 MB/s, 90.5 MB/s, and 72.5 MB/s for HVIA-GE, M-VIA, and TCP/IP, respectively. Therefore, HVIA-GE attains 24% and 55% higher bandwidth than M-VIA and TCP/IP, respectively.

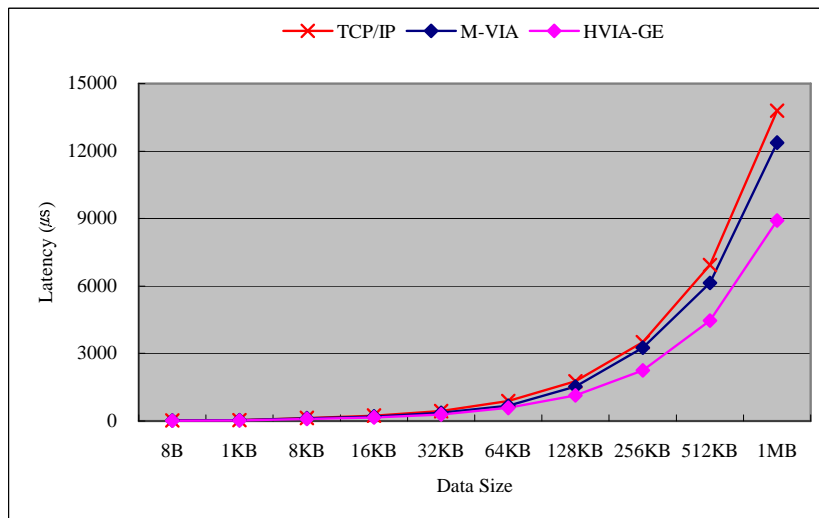


Fig. 5. Latency comparison

4.1.2 Small message transfer

Figure 7 shows the performance of send/receive operations for small messages, which are typical in cluster systems. The results of the RDMA operations were not included because the speed difference between send/receive and RDMA was quite small. As a result of having hardware doorbell, address translation using on-board SDRAM, pipeline data movement through the FIFOs, and fast

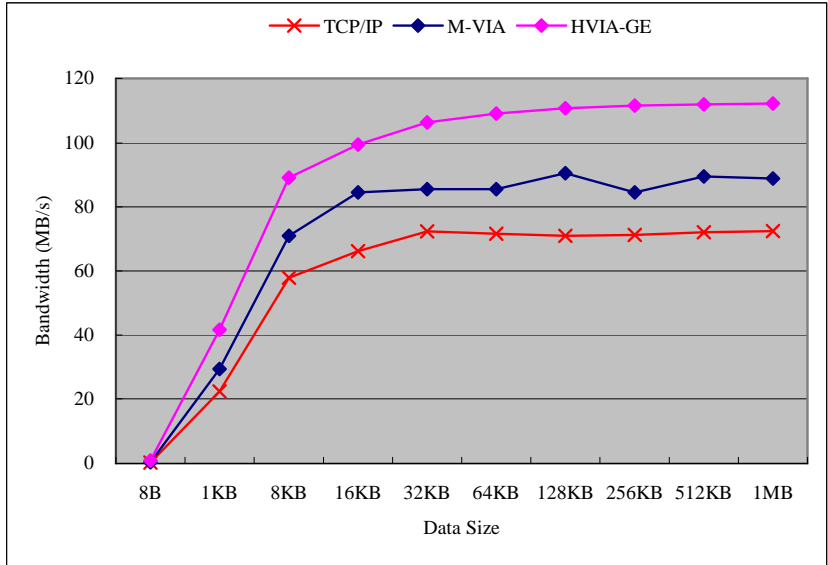


Fig. 6. Bandwidth comparison

completion mechanisms, the latency was less than $100 \mu s$ when the data size was below 10 KB, and the performance was 1.5-3.5 times faster than M-VIA or TCP/IP.

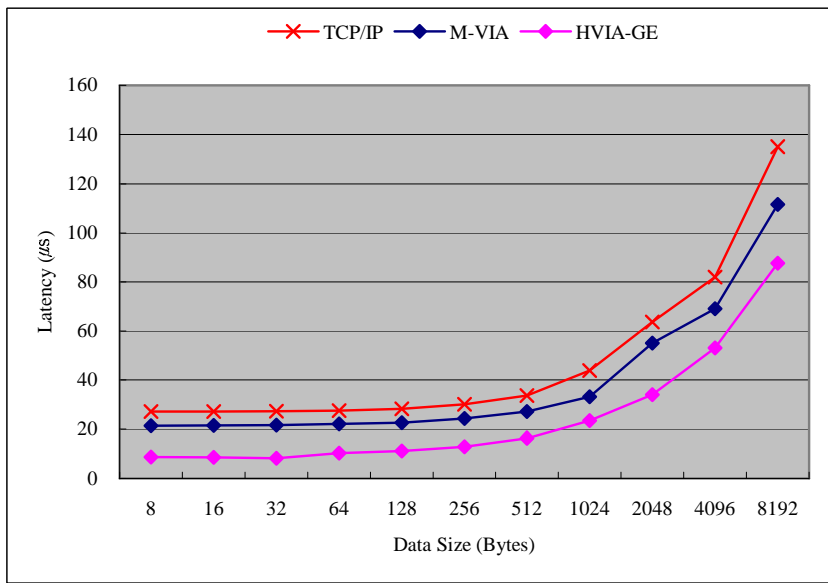


Fig. 7. Latency for small message transfer

4.2 Performance analysis of HVIA-GE

4.2.1 Latency analysis

The minimum latency analysis of HVIA-GE is shown in Figure 8. This analysis is based on the send/receive mechanism with the Address Prefetch Buffer. First, the time for posting a descriptor to the Send/Receive Queue (`VipPostSend/VipPostRecv`) is $0.9 \mu\text{s}$. In the sender, the processing time required by the VIA Protocol Engine and GEC is approximately $2.4 \mu\text{s}$. During this period, the VIA Protocol Engine reads a descriptor, calculates the physical address and reads the user buffer. Then, GEC moves the data to the Tx buffer and notifies the MAC to transmit the data, and at the same time, the VIA Protocol Engine processes the completion mechanism. The data transmission time from the Tx Buffer of the sender to the Rx Buffer of the receiver is approximately $3 \mu\text{s}$. In the receiver, the time required for the VI descriptor generation and address translation is not counted because those are performed at the same time the sender creates the VI descriptor. The time required to transfer data from Rx Buffer via the GEC and the VIA Protocol Engine to the host PCI bus is approximately $1 \mu\text{s}$, and completion time of a receive transaction is less than $1 \mu\text{s}$. If the address prefetch buffer is not used, the address translation time in the receiver will be included in the latency. In our implementation, the HVIA-GE card performs almost all of the send/receive processing; therefore, that the time spent in the host is minimized.

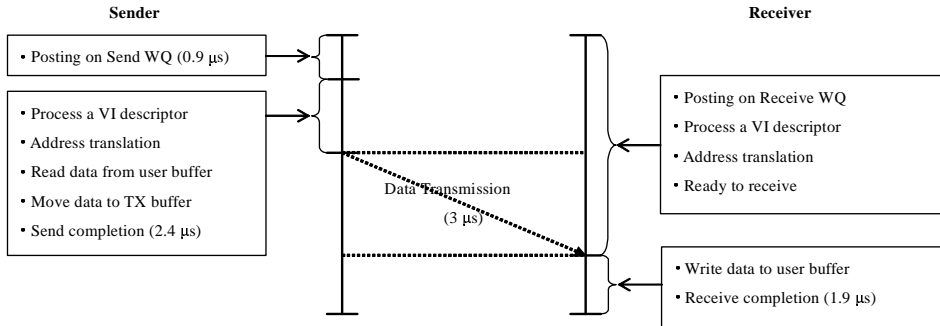


Fig. 8. Latency analysis

4.2.2 Send/receive vs. RDMA

Performing address translation using SDRAM on HVIA-GE takes approximately 450 ns . If the address translation table is located on the host memory, the time for address translation is slower by approximately a factor of three. In addition, the Address Prefetch Buffer further reduces latency in a send/receive transaction. Figures 9 and 10 show the effect of having the Address Prefetch Buffer. Figure 9 shows the difference in latency between RDMA write and

send/receive with and without the Address Prefetch Buffer. The average latency difference between an RDMA and a send/receive with and without the address prefetch buffer is $0.51 \mu\text{s}$ and $1.52 \mu\text{s}$, respectively. Although these differences are small for large messages, they are significant for small message transfers. Figure 10 shows the performance ratio compared with RDMA. In general, RDMA is faster than send/receive, but with the Address Prefetch Buffer send/receive achieves performance similar to RDMA.

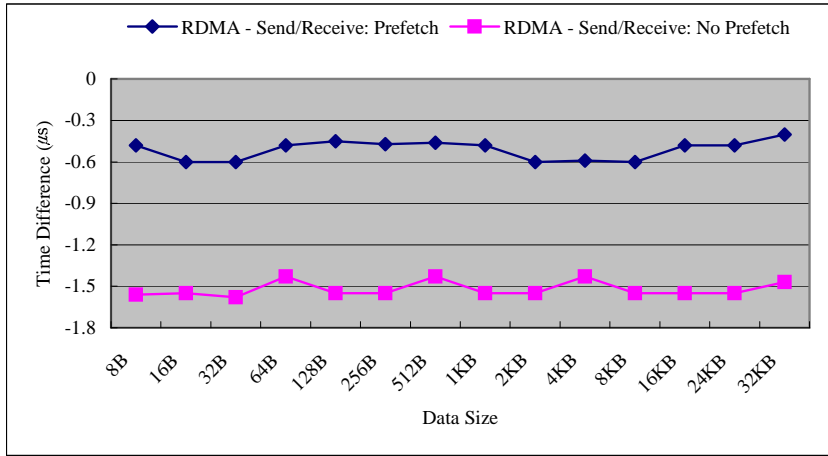


Fig. 9. Latency differences

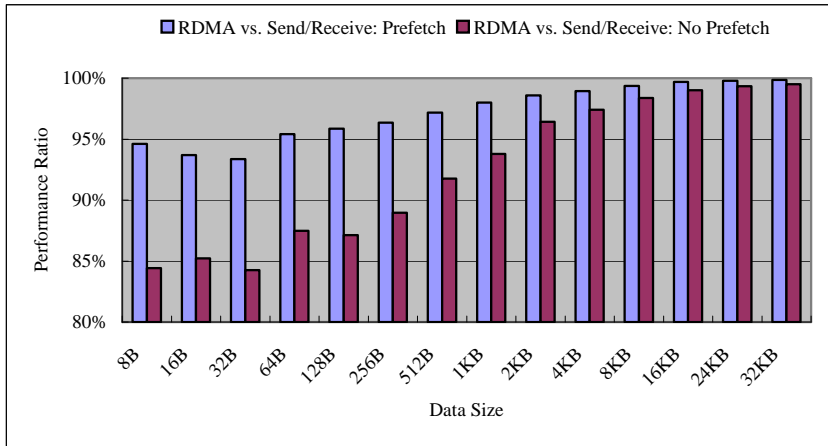


Fig. 10. Performance ratio

4.2.3 Blocking vs. Polling

Figure 11 compares the latency of the two completion mechanisms: blocking and polling. Latencies results for message sizes above 8 KB were excluded because they were very similar. In case of blocking, the minimum latency is $14.1 \mu\text{s}$ and the average latency difference with polling is $5.2 \mu\text{s}$, due to overheads for system calls, context switching, and interrupt processing. If small

message transactions are used frequently, polling will be advantageous compared to blocking in terms of latency. On the other hand, blocking is preferred for larger messages because the performance is similar but CPU utilization is greatly reduced.

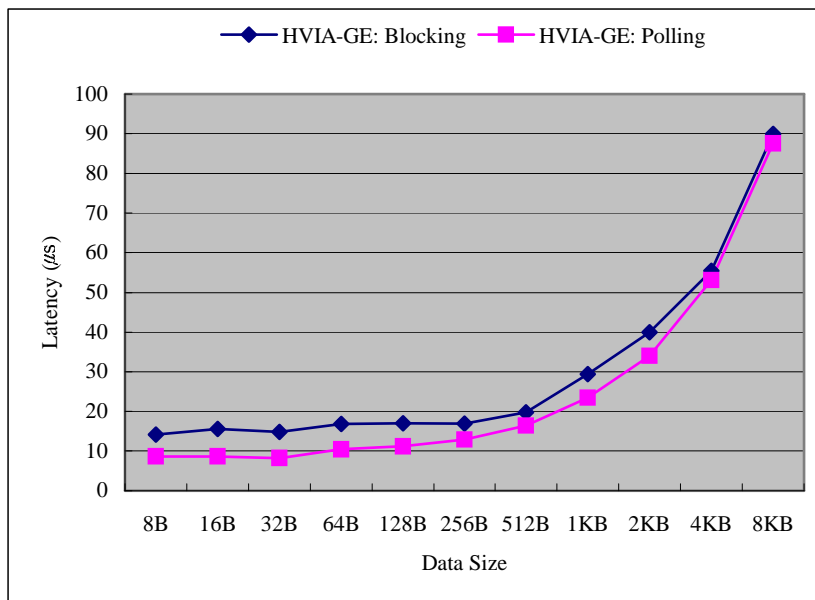


Fig. 11. Latency: Blocking vs. Polling

4.2.4 CPU utilization

Figure 12 shows the CPU utilization results for HVI A-GE with blocking, M-VIA and TCP/IP. The elements that influence CPU utilization are the CPU's jobs, such as data copy, context switching, system calls and other tasks for the CPU. CPU utilization for TCP/IP increases as data size grows, and becomes saturated at 55%-60%. In the case of M-VIA, CPU utilization is almost 100% when data size is below 4 KB, and saturated near 30%. This is because M-VIA calls the polling function (`VipSendDone`) 50,000 times before suspending the send/receive process [31]. For HVI A-GE, utilization decreases and saturates to below 1% when data size increases to about 10 KB. As shown in Figure 8, almost all the task of a send/receive is processed in the HVI A-GE card. The only portions that require CPU cycles are posting send/receive on WQ and handling send/receive completion. Moreover, an RDMA write does not necessarily require an explicit `VipPostRecv` in the receiver, therefore an RDMA write is superior to a send/receive in terms of CPU utilization.

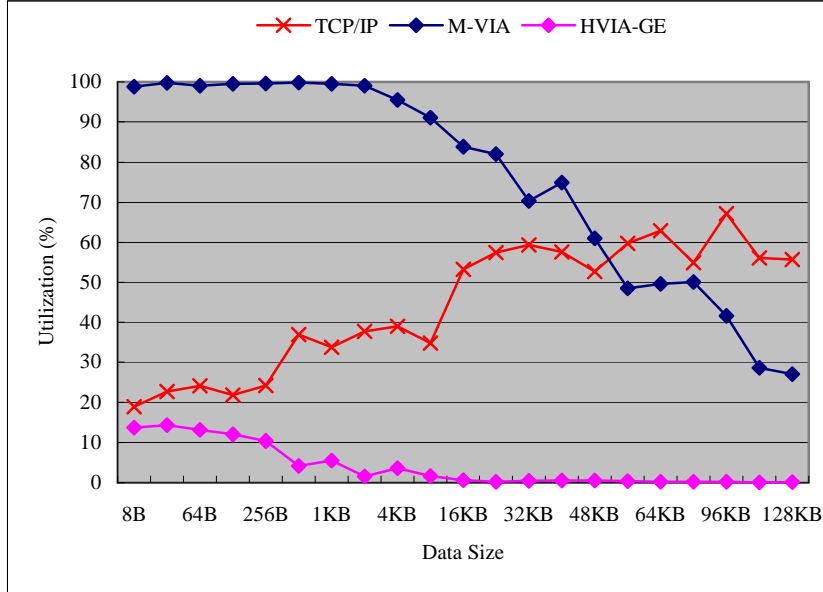


Fig. 12. CPU Utilization

5 Conclusions and Future Work

This paper presented the design and performance of HVIA-GE that implements the VIA protocol in hardware. The HVIA-GE card contains the VIA Protocol Engine and a Gigabit Ethernet Controller, and supports virtual-to-physical address translation, doorbell, RDMA write, and send/receive completion operations completely in hardware without intervention from the kernel. In particular, the ATT is stored in the local memory on the HVIA-GE card and the VIA Protocol Engine directly and efficiently controls the address translation process. In addition, the Address Prefetch Buffer reduces the time for the address translation process at the receiver.

Our experiment with HVIA-GE showed a minimum latency of $8.2 \mu\text{s}$, and a maximum bandwidth of 112.1 MB/s. These results indicate that the performance of HVIA-GE is much better than that of M-VIA, which is a software implementation of VIA, and also better than that of ServerNet II, and comparable with cLAN. In addition, RDMA write is superior to send/receive not only in terms of latency and bandwidth, particularly with data sizes below 10 KB, but also in terms of CPU utilization for applications that communicate large data messages.

Currently, the HVIA-GE card is being tested with a video streaming server and we are developing a zero-copy file transfer mechanism with RDMA using the HVIA-GE. Unlike Direct Access File System (DAFS) [26][27], a user-level file system based on VIA, our mechanism will provide simple and VIA-like file transfer primitives without modifying the file system and is easily

adaptable and reconfigurable to HVIA-GE. Also, the PCI interface of HVIA-GE can be further upgraded to support PCI-X and PCI Express. With these enhancements, the HVIA-GE card will be improved and practical for use in Gigabit Ethernet-based cluster systems.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments. This work was supported by "Research Center for Logistics Information Technology (LIT)" hosted by the Ministry of Education & Human Resources Development in Korea.

References

- [1] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauer, Active Messages: a Mechanism for Integrated Communication and Computation, in: 19th International Symposium on Computer Architecture, May 1992, pp. 256-266
- [2] S. Pakin, M. Lauria, and A. Chien, High Performance Messaging on Workstation: Illinois Fast Message (FM) for Myrinet, in: Proc. of Supercomputing'95, December 1995, pp. 1528-1557
- [3] T. Von Eicken, A. Basu, V. Buch, and W. Vogels, U-NET: A User Level Network Interface for Parallel and Distributed Computing, in: Proc. of the 15th ACM Symposium on Operating Systems Principles (SOSP), December 1995, pp. 40-53
- [4] Virtual Interface Architecture Specification, <http://www.viarch.org/>
- [5] P. Bozeman and B. Saphir, A Modular High Performance Implementation of the Virtual Interface Architecture, in: Proc. of the 2nd Extreme Linux Workshop, June 1999
- [6] P. Buonadonna, A. Geweke, and D.E. Culler, An Implementation and Analysis of the Virtual Interface Architecture, in: Proc. of the Supercomputing'98, November 1998, pp. 7-13
- [7] <ftp://ftp.compaq.com/pub/supportinformation/papers/tc000602wp.pdf>, Compaq ServerNet II, June 2000
- [8] Emulex Corporation, Hardware-based (ASIC) implementation of the Virtual Interface standard, <http://www.emulex.com/products/legacy/index.html>

- [9] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, W. Su, Myrinet-A Gigabit Per Second Local Area Network, in: IEEE Micro, February 1995, pp. 29-36
- [10] IEEE Standard for Scalable Coherent Interface, IEEE Std 1596-1992, IEEE Computer Society, August 1993
- [11] M. Banikazemi, J. Liu, S. Kutlug, A. Ramakrishna, P. Sadayappan, H. Sah, and D. K. Panda, VIBe: A Micro-benchmark Suite for Evaluating Virtual Interface Architecture (VIA) Implementations, in: Int'l Parallel and Distributed Processing Symposium (IPDPS), April 2001, pp. 24
- [12] M. Baker, P. A. Farrell, Hong Ong, and S. L. Scott, VIA Communication Performance on a Gigabit Ethernet Cluster, in: Proc. of the Euro-Par 2001, August 2001, pp. 132-141
- [13] H. Ong and P. A. Farrell, Performance Comparison of LAM/MPI, MPICH, and MVICH on a Linux Cluster connected by a Gigabit Ethernet Network, in: Proc. of the 4th Annual Linux Showcase & Conference, October 2000, pp. 353-362
- [14] M. Banikazemi, B. Abali, and D. K. Panda, Comparison and Evaluation of Design Choices for Implementing the Virtual Interface Architecture (VIA), in: Fourth Int'l Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing (CANPC '00), January 2000, pp. 145-161
- [15] I. S. Yoon, S. H. Chung, B. Lee, and H. C. Kwon, Implementation and Performance Evaluation of M-VIA on AceNIC Gigabit Ethernet Card, in: Proc. of the Euro-Par 2003, August 2003, pp. 995-1000
- [16] M. Banikazemi, B. Abali, L. Herger, and D. K. Panda, Design Alternatives for Virtual Interface Architecture (VIA) and an Implementation on IBM Netfinity NT Cluster, in: Journal of Parallel and Distributed Computing, Special Issue on Clusters, Special Issue on Cluster and Network-Based Computing, Vol. 61, No. 11, November 2001, pp. 1512-1545
- [17] M. Banikazemi, V. Moorthy, L. Hereger, D. K. Panda, and B. Abali, Efficient Virtual Interface Architecture Support for IBM SP switch-connected NT clusters, in: Proc. Of International Parallel and Distributed Processing Symposium, May 2000, pp. 33-42
- [18] InfiniBand Trade Association, <http://www.infinibandta.org/>
- [19] F. Seifert and W. Rehm, Proposing a Mechanism for Reliably Locking VIA Communication Memory in Linux, in: IEEE International Conference on Cluster Computing (CLUSTER'00), November 2000, pp. 225-232
- [20] A. Rubini, J. Corbet, Linux Device Driver, 2nd Edition, in: O'Reilly, June 2001
- [21] Intel Corporation, 82544 Gigabit Ethernet Controllers with Integrated PHY, <http://www.intel.com/design/network/products/lan/controllers/82544.htm>
- [22] http://www.plxtech.com/products/io_accelerators/default.asp, PLX Technology, PCI I/O Accelerators

- [23] <http://www.amd.com/>, Advanced Micro Devices, AMD Opteron 246 server processor
- [24] <http://www.intel.com/network/connectivity/products/>, Intel Corporation , Intel PRO/1000 MT Server Adapter
- [25] M. Blumrich, C. Dubnicki, E. W. Felten, K. Li, and M. R. Mesarina, Virtual-Memory-Mapped Network Interface, in: IEEE Micro, February 1995, pp. 21-28
- [26] DAFS Collaborative. Direct Access File System Protocol, Version 1.0, September 2001, <http://www.dafscollaborative.org>
- [27] A. Fedorova, M. Seltzer, K. Magoutis, S. Addetia, Application performance on the Direct Access File System, in: ACM SIGSOFT Software Engineering Notes, Proceedings of the 4th international workshop on Software and Performance, vol 29, January 2004, pp. 84-93
- [28] Mellanox Technologies, <http://www.mellanox.com/products/products.html>
- [29] Chris Eddington, InfiniBridge: an InfiniBand channel adapter with integrated switch, in: IEEE Micro, Vol 22, Issue 2, March-April, 2002
- [30] Hermann Hellwagner and Matthias Ohlenroth, VI architecture communication features and performance on the Gigabit cluster LAN, in: Future Generation Computer Systems, Vol. 18, Issue 3, January 2002, pp. 421-433
- [31] M-VIA Core Release 1.2, <http://old-www.nersc.gov/research/FTG/via/>