HVIA-GE: A Hardware Implementation of Virtual Interface Architecture Based on Gigabit Ethernet

Sejin Park[1], Sang-Hwa Chung[1], In-Su Yoon[1], In-Hyung Jung[1], So Myeong Lee[1], and Ben Lee[2]

[1] Department of Computer Engineering
Pusan National University, Pusan, Korea
{sejnpark,shchung,isyoon,jung,smlee3}@pusan.ac.kr
[2] School of Electrical Engineering and Computer Science
Oregon State University, USA
benl@eecs.orst.edu

Abstract. This paper presents the implementation and performance of the HVIA-GE card, which is a hardware implementation of the Virtual Interface Architecture (VIA) based on Gigabit Ethernet. VIA is a user-level communication interface for high performance PC clustering. The HVIA-GE card is a 32-bit/33MHz PCI adapter containing an FPGA for the VIA Protocol Engine (VPE) and a Gigabit Ethernet chip set to construct a high performance physical network. HVIA-GE performs virtual-to-physical address translation, doorbell, 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 HVIA-GE card, and the VPE efficiently controls the address translation process by directly accessing the ATT. As a result, the communication overhead during send/receive transactions is greatly reduced. Our experimental results show a minimum latency of 12.2 μs, and a maximum bandwidth of 96.3 MB/s. In terms of minimum latency, HVIA-GE performs 4.7 times and 9.7 times faster than M-VIA, a software implementation of VIA, and TCP/IP, respectively, over Gigabit Ethernet. In addition, the maximum bandwidth of HVIA-GE is 52% and 60% higher than M-VIA and TCP/IP, respectively.

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, Fast Messages, U-Net, and VIA [8]. Among them, Virtual Interface Architecture (VIA) was proposed to standardize different features of

C. Aykanat et al. (Eds.): ISCIS 2004, LNCS 3280, pp. 198–206, 2004.

existing user-level protocols for System Area Network. VIA can be implemented in either software or hardware. The software implementations include M-VIA [3] and Berkeley VIA [4], and the hardware implementations include ServerNet II [6], and cLAN [5].

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 LAN and WAN, it has an advantage in terms of cost when compared with proprietary high performance networks, such as Myrinet and SCI. 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. Currently, there are a number of efforts to implement software versions of VIA based on Gigabit Ethernet using either M-VIA or Berkeley VIA [2],[1],[7],[9]. Meanwhile, Tandem/Compaq developed ServerNet II, a hardware version of VIA using Gigabit Ethernet as a physical network. ServerNet II uses its own switch, which supports wormhole routing with 512-byte packets, to connect cluster of nodes. ServerNet II shows a minimum latency of 12 μs for 8-byte data and a bandwidth of 92MB/s for 64 KB data using RDMA 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 implemented as a hardware VIA, and shows a minimum latency of 7 μs and a maximum bandwidth of 110MB/s. Although cLAN shows better performance than ServerNet II, it is based on an expensive proprietary network, similar to Myrinet and SCI.

This paper presents the design and implementation of HVIA-GE, which is a *H*ardware implementation of *VIA* based on *G*igabit *E*thernet. HVIA-GE is a PCI plug-in card based on 33MHz/32-bit PCI bus. An FPGA was used to implement the VIA Protocol Engine (VPE) and a Gigabit Ethernet chip set was used to connect the VPE 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 VPE efficiently performs the virtual-to-physical address translation. The PCI logic was directly implemented on the FPGA instead of 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.

2 VIA Overview

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. 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 performs 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.

3 Implementation of HVIA-GE

Our implementation of the major components of VIA is highlighted as follows. First, WQ is stored in the host memory but the VI descriptors that are currently being processed are copied and stored in the HVIA-GE card 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. 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 for efficient address translation. The VPE controls the address translation process directly based on the ATT.

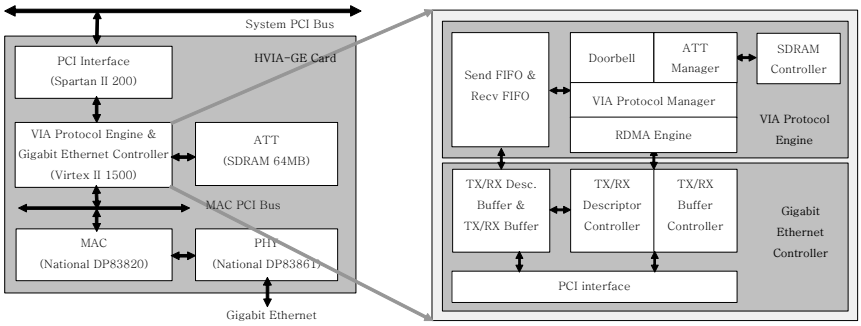


Fig. 1. HVIA-GE Card block diagram

Figure 1 shows the block diagram of the HVIA-GE card, which is a network adapter based on 33MHz/32-bit PCI bus. The PCI interface logic, VPE, the SDRAM controller, and the Gigabit Ethernet Controller (GEC) are all implemented using FPGA running at 33 MHz. The PCI interface logic is implemented directly on the FPGA, rather than using a commercial chip, such as PLX9054, to minimize the latency of the DMA initialization. National Semiconductor's MAC (DP83820) and PHY (DP83861) are used to connect the card to Gigabit Ethernet. 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

As shown in Fig. 1, VPE and GEC are the core modules of HVIA-GE. VPE 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 VipRegisterMem, which is the VIPL function used to register 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. For example, a 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, the 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. 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 VPE, 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 the MAC of the start of a transmission. Then, the MAC reads the packet from the Tx Buffer and transfers it to the PHY.

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, a user buffer area must be pinned down when it is registered so that it is not swapped out during

send/receive operations. In our implementation, one of the Linux kernel's features, `kiobuf`, is used to pin down the user buffer. The virtual address and the corresponding physical address of the user buffer obtained during the pin down process are saved on ATT.

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 the physical pages of the user buffer, the virtual address and the size of the first page, and ATT Level 2 pointer. ATT Level 2 stores the physical addresses (4-byte each) of all the allocated pages for the corresponding user buffer. 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 the ATT supports 1024 VIs and each VI uses one user buffer, then the 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, the SDRAM can support more than 3 million user buffers. Therefore, the capacity of the 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 VPE reads the corresponding ATT Level 1 entry using the memory handle. This requires only one SDRAM access to read the entire entry, which is 24 bytes, 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.

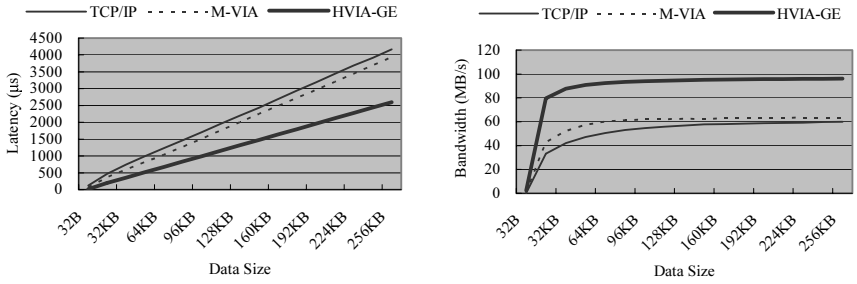
Unlike the approach described here, it is also possible to implement ATT on the host memory. In this case, VPE has to access ATT via DMA reads. Although this method has an advantage in terms of the hardware cost, it takes about 3 times longer to access the ATT on the host memory than on SDRAM of the 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.

4 Experimental Results

The performance of the HVIA-GE card was evaluated using two 800 MHz Pentium III PCs with 32-bit/33MHz/PCI bus. The PCs were running RedHat 7.2 with Linux kernel 2.4. Also, for comparison purposes, the performances of TCP/IP and M-VIA were measured using AceNIC's Gigabit Ethernet card. The latency and bandwidth of HVIA-GE were measured by using a ping-pong program developed using VIPL. The performance of M-VIA was measured using the vnettest program included with the M-VIA distribution. 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



(a) Latency (b) Bandwidth

Fig. 2. Latency and Bandwidth Comparisons

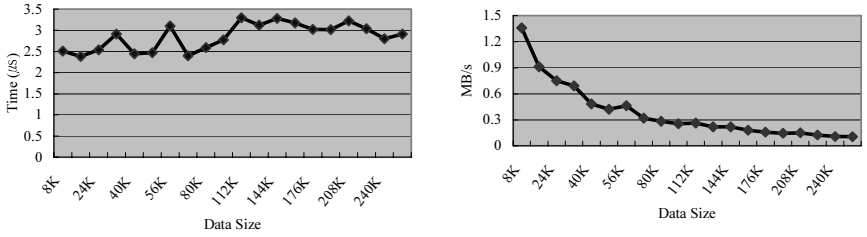
Fig. 2 shows the latencies and the bandwidths results of HVIA-GE, M-VIA, and TCP/IP with the Ethernet MTU size of 1,514 bytes. The latency reported is one-half the round-trip time and the bandwidth is the total message size divided by the latency. The latency and bandwidth of HVIA-GE are measured using the RDMA write on a single VI channel. The minimum latency results of HVIA-GE, M-VIA, and TCP/IP for 4 bytes of user data are 12.2 μs, 57.6 μs, and 117.9 μs, respectively. Thus, the minimum latency of HVIA-GE is 4.7 and 9.7 times lower than M-VIA and TCP/IP, respectively. The maximum bandwidth results for 256 KB of user data are 96.3 MB/s, 63.5 MB/s, and 60 MB/s for HVIA-GE, M-VIA, and TCP/IP, respectively. Thus, HVIA-GE achieves 50% and 59% higher bandwidth than M-VIA and TCP/IP, respectively.

The minimum latency of H-VIA can be analyzed as follows. In the sender, the processing time required by the VPE and GEC is approximately 5.6 μs, and

the data transmission time from Tx Buffer of the sender to Rx Buffer of the receiver is approximately 4.1 μs. In the receiver, the time required to transfer data from Rx Buffer via GEC and the VPE to the PCI bus is approximately 1.5 μs. Thus, in our implementation, the time spent in the host to call the send VIPL function, update WQ, and post the corresponding transaction is less than 1 μs.

4.2 Performance Comparison of Send/Receive and RDMA Write on HVIA-GE

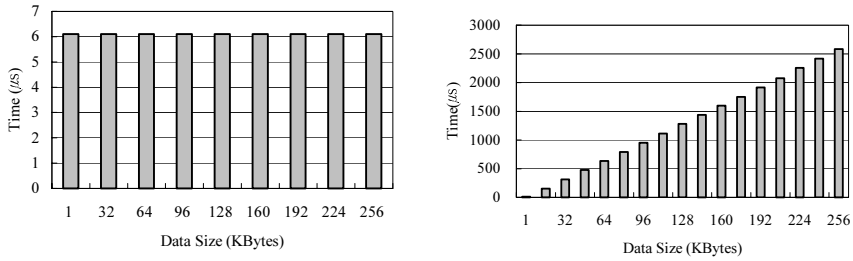
Latency and Bandwidth. Fig. 3 shows the performance difference between RDMA write and send/receive. As shown in Fig. 3-a, the latency difference is on average 2.9 μs regardless of data size. Fig. 3-b represents the difference in bandwidth. These improvements are obtained because RDMA write does not require a VI descriptor to be created and posted to WQ at the target node. Thus, the completion mechanism that sets the done bit in the VI Descriptor's status field is also not necessary at the target node. Although the improvement is not significant for large data, it is meaningful to the applications that communicate using data size below 10 to 20 KB.



(a) Difference in Latency (b) Difference in Bandwidth

Fig. 3. Differences in Latency and Bandwidth

CPU Utilization. Unlike the RDMA write, a send/receive requires some VIPL functions such as VipPostRecv and VipRecvWait at the receiver. VipPostRecv generates a VI descriptor, posts it to WQ and waits for HVIA-GE to read it. VipRecvWait waits for the message to arrive and completes the transaction when the message is DMAed to the user buffer. Fig. 4 shows the CPU times spent for VipPostRecv and VipRecvWait. The CPU time spent for VipPostRecv is about 6.1 μs regardless of data size. However, the CPU time spent for VipRecvWait increases in proportion to the data size. Thus, this represents a significant portion of the latency for applications that communicate using large data. In the case of RDMA write, VipPostRecv and VipRecvWait are not necessary, thus the RDMA write is superior to the send/receive in terms of CPU utilization.



(a) CPU time for VipPostRecv (b) CPU time for VipRecvWait

Fig. 4. CPU time in Receive Node.

5 Conclusions and Future Work

In this paper, we presented the design and performance of HVIA-GE that implements the VIA protocol in hardware. The HVIA-GE card contains VPE and GEC, 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, ATT is stored in the local memory on the HVIA-GE card and VPE directly and efficiently control the address translation process.

Our experiment with HVIA-GE shows a minimum latency of 12.2 μs, and a maximum bandwidth of 96.3 MB/s. These results indicate that the performance of HVIA-GE is much better than M-VIA and TCP/IP, and is comparable with that of ServerNet II. If applications with frequent send/receive transactions are executed, the performance of HVIA-GE will be better than that of ServerNet II because of the hardware implementation of ATT. In addition, it is easier to construct a low-cost, high performance cluster system compared to ServerNet II because HVIA-GE is connected to Gigabit Ethernet using the general Gigabit Ethernet switches developed for LAN. In our experiment with HVIA-GE, the RDMA write is better than the send/receive in terms of latency and bandwidth, particularly with data size below 10 to 20 KB. In addition, the RDMA write is superior to the send/receive in terms of CPU utilization for applications that communicate using large data.

As a future work, the HVIA-GE card will be further tested with real applications such as video streaming server. We also plan to make the HVIA-GE card support 64-bit data width. This means that the HVIA-GE card will support the 64-bit/66MHz PCI bus for the system and the MAC, and all the data paths of VPE and SDRAM will be 64 bits. The PCI interface of the HVIA-GE card can be further upgraded to support PCI-X and PCI Express. With these enhancements, the performance of the HVIA-GE card will be significantly improved.

References

1. Baker, M., Farrell, P.A., Ong H., Scott, S.L.: VIA Communication Performance on a Gigabit Ethernet Cluster. In: Proc. of the Euro-Par 2001. (2001)
2. Banikazemi, M., Liu, J., Kutlug, S., Ramakrishna, A., Sadayappan, P., Sah, H., Panda, D.K.: VIBe: A Micro-benchmark Suite for Evaluating Virtual Interface Architecture (VIA) Implementations. In: Int'l Parallel and Distributed Processing Symposium (IPDPS). (2001)
3. Bozeman, P., Saphir, B.: A Modular High Performance Implementation of the Virtual Interface Architecture. In: Proc. of the 2^{nd} Extreme Linux Workshop. (1999)
4. Buonadonna, P., Geweke, A., Culler. D.E.: An Implementation and Analysis of the Virtual Interface Architecture. In: Proc. of the Supercomputing'98. (1998)
5. Emulex Corporation: Hardware-based (ASIC) implementation of the Virtual Interface (VI) standard. `http://www.emulex.com/products/legacy/index.html`
6. `ftp://ftp.compaq.com/pub/supportinformation/papers/tc000602wp.pdf`
7. Ong H., Farrell, P.A.: Performance Comparison of LAM/MPI, MPICH, and MVICH on a Linux Cluster connected by a Gigabit Ethernet Network. In: Proc. of 4th Annual Linux Showcase & Conference. (2000)
8. Virtual Interface Architecture Specification. `http://www.viarch.org/`
9. Yoon I.S., Chung, S.H., Lee, B., Kwon, H.C.: Implementation and Performance Evaluation of M-VIA on AceNIC Gigabit Ethernet Card. In: Proc. of the Euro-Par 2003. (2003)