An Efficient Implementation of Virtual Interface Architecture using Adaptive Transfer Mechanism on Myrinet

Jung-Lok Yu, Moon-Sang Lee, Seung-Ryoul Maeng
Division of Computer Science of Department of Electrical Engineering and Computer Science, Korea Advanced Institute Science and Technology
{jlyu, mslee, maeng}@calab.kaist.ac.kr

Abstract

The user-level communication is investigated by many researches, in order to resolve the performance degradation of cluster systems due to inefficient communication protocols. It removes the kernel intervention from the critical communication path. Recently, Intel, Microsoft and Compaq introduce the Virtual Interface Architecture (VIA), a standard for user-level communication. However, the existing VIA implementation shows low performance in transferring small messages, because it uses a single mechanism to transfer messages without regard to their message size. In this paper, we implement a high performance VIA, KVIA (Kaist VIA). KVIA, based on descriptor and message size, dynamically selects a proper transfer mechanism. This implementation effectively handles not only large messages but also small messages. Thus, it can be better applied to the systems that frequently use small messages (e.g., lock protocols for software distributed shared memory). The performance of KVIA is reported using round-trip latency and one-way bandwidth. Our results show the round-trip latency of 40 micro-seconds and the maximum one-way bandwidth of 950 Mbits per second, which is about 74% of Myrinet link’s peak bandwidth.

1. Introduction

Recently, there is rapid development in network and microprocessor technologies. Based on this development, the clustering that connects high performance PCs and workstations by high-speed networks have been appeared. The cluster system becomes prevalent to substitute MPP (Massively Parallel Processor) in parallel computing. Also, it will be used to build internet-based servers, for example, Web servers. However, the performance of cluster system is critically limited by poor traditional communication protocols (e.g., TCP/IP), because they show high overheads in context switching, memory copy and buffer management.

User-level communications are proposed extensively in order to avoid the transfer latency and seek the maximum utilization of cluster systems [1,2]. The underlying idea of user-level communications is to bypass kernel in transferring data. It requires a mechanism for user processes to access network interfaces directly. In this way, user processes can transfer data with low latency and make efficient use of system resources without context switching and memory copy. A lot of protocols for the user-level communications have been implemented: AM (Active Message)[3], FM (Fast Message)[4], U-Net[5], Trapeze[6], Myricom GM[7], BIP (Basic Interface for Parallelism)[8], VMMC (Virtual Memory Mapped Communication)[9], and so on.

VIA (Virtual Interface Architecture)[10,11,12] is the industry standard for the above user-level communication protocols. Most parallel applications, Web server control programs, MPI (Message Passing Interface) and TCP socket APIs (Application Programming Interface) can be implemented using VIA standard interface. Considering the small size of locks in parallel applications, data exchanged between clients and Web servers, and control messages in MPI, it is very important to implement VIA protocol that can transfer small messages efficiently, which is easily overlooked in most existing VIA implementations. Berkeley-VIA [13] transfers data using a single mechanism without regard to data size. It is highly ineffective to transfer small sized messages using the single mechanism, even by the sequential control flow. In this paper, we implement a high performance VIA, Kaist VIA (KVIA), making use of two transfer mechanisms with different characteristics. Based on descriptor and data size, KVIA adaptively selects a proper transfer mechanism during the run-time. Thus, our implementation effectively handles both kinds of message size.

The remainder of paper is as follows: Section 2 reviews the fundamental mechanics of VI Architecture. In Section 3, we describe the design and implementation of KVIA in detail. Section 4 reports the communication

\footnote{This project is supported by the National Research Lab Grant 2-407.}
2. VIA fundamental

VIA consists of 4 parts (see Figure 1). First, VI (Virtual Interface) not only is a endpoint in point-to-point VIA communication protocol, but also provides user processes with a virtual path to network device. Two work queues (send and receive queue) and two doorbells (send and receive doorbell) exist within a VI. Second, VI provider is composed of a kernel agent and a network controller. VI provider executes device open/close, VI creation/destruction, connection setup/teardown and memory registration/deregistration, etc. Third, VI consumer points to user-side processes by interfacing VIs and user processes. Fourth, completion queue allows a VI consumer to coalesce notification of request completions of multiple VIs in a single location.

Figure 1. VI architectural model

Data buffers and the work queues themselves are allocated in an area of the host's memory. These memory regions are pinned in physical memory and are called registered memory in VIA terminology. VIA is a connection-oriented protocol. The endpoint must be connected to a remote endpoint before packets can be sent. VIA provides two communication models: primitive send/receive model and RDMA (Remote Direct Memory Access) read/write model. User process initiates data communication by constructing an appropriate descriptor in work queue and ringing the appropriate doorbell on the network interface card (NIC). VIA provides two descriptor completion models: Work Queue model and Completion Queue model, and also supports three levels of communication reliability at the NIC level: Unreliable Delivery, Reliable Delivery and Reliable Reception.

Berkeley-VIA is an implementation of VIA. Berkeley-VIA is developed using Myrinet[14] M2-PCI-32 NIC. And Berkeley-VIA supports primitive send/receive model and unreliable delivery model. Because Myrinet M2-PCI-32 NIC does not support hardware doorbell mechanism, Berkeley-VIA's doorbells are implemented by polling single memory location. Such a doorbell mechanism generates a lot of load for Myrinet local bus and deteriorates the communication performance. As Berkeley-VIA's MCP completes each transaction in strict sequence, making no attempt to pipeline the available data-transfer DMA engines, this protocol doesn't fully exploit the system resources. In addition, Berkeley-VIA implementation has lower performance in sending or receiving small messages. This is because it uses a uniform mechanism to transfer messages without regard to their message size.

3. KVIA design and efficient transfer mechanism

3.1. KVIA structure

The implementation of KVIA consists of a library as VI user agent, KVIA kernel agent and MCP that controls the operation of Myrinet NIC. Figure 2 shows KVIA structure.

Figure 2. KVIA structure

KVIA user library calls system functions following the types of user process requests, or interacts directly with MCP for data transmission/reception. KVIA kernel agent initializes internal data structures and informs MCP of user requests using MCP control commands. Interrupt and I/O mapped buffer are the methods, with which kernel agent and MCP communicate. Because KVIA user library operates in user address space, sending or receiving data are executed without kernel control. MCP requires the means to translate virtual to physical address, to report errors and connection setup/teardown to kernel agent, and these requirements are met by interrupt. KVIA kernel agent executes registered interrupt service routines, and then informs MCP of the result about its request.

3.2. Doorbell mechanism in KVIA

VI is the basic unit of communication in VIA. In KVIA, VI locates in host memory to reduce I/O bus load and save Myrinet SRAM, a limited resource. Two doorbells in VI are the paths to control Myrinet NIC for transferring data. A user process informs MCP that there are data to transfer by ringing doorbell. KVIA uses a hardware doorbell
mechanism supported in Myrinet M2L-PCI64A-4. Figure 3 shows the doorbell mechanism in KVIA.

![Figure 3. KVIA hardware doorbell mechanism](image)

A user process can create maximum 512 VIs in a protected 4KB page. KVIA implements doorbell by mapping doorbell region, 8MB ~ 16MB, in Myrinet address space with the virtual address space assigned for user process without the interference of other user processes. A unique VI can be easily classified because of the static doorbell offset in a certain page. After mapping doorbell region with user space, if a doorbell is written in the mapped doorbell region, the doorbell is automatically recorded and logged in SRAM FIFO doorbell queue according to the hardware doorbell mechanism. Consequently, MCP detects doorbell minimizing the load of local bus in Myrinet NIC.

3.3. Adaptive transfer mechanism

VIA requires a descriptor and data movement between the host memory assigned to the user process and the memory of NIC for sending or receiving data. Berkeley-VIA uses the host-side DMA engine in Myrinet NIC to transfer data. However, because of the large start-up latency of DMA engine, small data transmission using DMA can cause the large latency. To provide user process as small latency as possible, we apply an adaptive transfer mechanism within KVIA. KVIA adaptively selects proper transfer mechanism based on descriptor and data size during run-time.

3.3.1. Environment setup.

KVIA sets a basic environment to provide adaptive transfer mechanism to user process. Such a setup is made transparently as user process attempts to open Myrinet NIC. When a NIC open request is taken by KVIA user library, the library calls a mmap system call to assign mapped buffers for adaptive transfer mechanism. After mapping process between the host memory and Myrinet SRAM, KVIA kernel agent not only returns the address of mapped buffer, but also informs MCP of memory mapping event. Figure 4 describes this mapping process.

![Figure 4. Mapping process between node memory and Myrinet SRAM](image)

KVIA user library obtains total 4 buffer areas after mapping process: the areas for send descriptor, receive data, to send and to receive, and these buffers are shared among VIs that are created by a user process.

3.3.2. Data transmission.

After building a send descriptor and data in the registered memory, a user process calls `VipPostSend` API to transfer data. KVIA user library adaptively selects how the data is moved into Myrinet SRAM by examining the size of posted descriptor and data. If their size is more than a selected threshold, the descriptor and data are moved into SRAM by the host-side DMA engine in Myrinet NIC. Otherwise, they are moved into SRAM by writing to the mapped buffer area described in 3.3.1. Each descriptor and data threshold is set to 64bytes and 256bytes, which values are dynamically selected by comparing Programmed I/O(PIO) throughput with DMA throughput during loading period of KVIA kernel agent and MCP.

When a user process builds a send descriptor and data whose size is less than the selected threshold values, `VipPostSend` API does copy the descriptor and data to the mapped buffer area. And `VipPostSend` API rings PIO doorbell (see Figure 5) to inform MCP of small data transmission by writing it into the mapped doorbell buffer. A PIO doorbell includes the information where the descriptor and data is written in SRAM. When MCP detects the PIO doorbell from user process, a descriptor and data are already located in Myrinet SRAM.

![Figure 5. PIO doorbell format](image)

When a user process builds them and their size is more than threshold, `VipPostSend` API examines the address of descriptor to be aligned with 64 bytes. After examination, it rings a DMA doorbell by the same way. Figure 6 shows the format of DMA doorbell. A DMA doorbell includes the information about the size and virtual address of descriptor. When MCP detects the DMA doorbell, MCP
extracts the virtual address and size of descriptor from the doorbell. Based on the extracted information from doorbell, MCP obtains the send descriptor using a DMA engine in Myrinet NIC, then based on the obtained descriptor, transfer data from the host memory to Myrinet SRAM.

Figure 6. DMA doorbell format

After MCP transfers the data into Myrinet link using network-side DMA engine in Myrinet NIC, in case of small sized data, MCP informs user process of the completion by writing the status in status field in the descriptor, and, in other case, uploading the descriptor using host-side DMA engine.

### 3.3.3. Data reception

If local user process is intended to receive data from remote process, it must build a receive descriptor in the registered memory region and post a receive doorbell. As KVIA only supports the lowest reliability, Unreliable Delivery, the receive doorbell should be preposted before packet is received from Myrinet link. Preposted receive doorbells are logged in Myrinet SRAM up to maximum 512.

As in the procedure of receiving data, threshold size is set to 64bytes for all of descriptor and data. If the size of receive descriptor and data to receive is less than the threshold, KVIA user library does copy the receive descriptor to mapped buffer area, resulting in the movement of it into Myrinet SRAM. After this step, KVIA user library rings PIO doorbell. When the size of data received from link is less than threshold value, MCP extracts the information of SRAM location where to write the received data from the logged PIO receive doorbell, and just records data into the extracted location. Afterward, as user process calls VipRecvDone API, KVIA user library copies the received data from the mapped buffer area to user process’s address space. Otherwise, if user process is intended to receive data which is larger than the threshold size, data reception is executed by the similar way to data transmission.

### 3.4. Parallel execution of DMA engines in Myrinet NIC

As describes in 3.3, Myrinet NIC has two DMA engines; one for transferring data between the host memory and Myrinet SRAM, the other between Myrinet SRAM and Myrinet link. Thus, to make full utilize of 1.28 Gbps link bandwidth, it is indispensable to parallelize these engines. KVIA supports the parallelization of DMA engines through double buffering, which means that each buffer is used to transfer data alternatively. Figure 7 shows the critical path of transferring data in KVIA.

To send data, user process builds a send descriptor and rings doorbell. On detecting the doorbell ringing, MCP transfers the send descriptor and data in order. After sending data to Myrinet link, MCP updates the status field in the send descriptor to inform user process of send completion. User process intended to receive data must prepost a receive descriptor and a receive doorbell before data is arrived at Myrinet SRAM. When data is received from Myrinet link, MCP transfers the receive descriptor according to the logged receive doorbell, and uploads the received data into user process’s memory. At that time, the status information is uploaded with data. KVIA executes the parallelization of DMA engines in two places; one is between TxData and Transfer which are using independent DMA engines, the other is between Receive and RxData. Through these parallelization, consequently, KVIA can hide the latency for the one of parallelized items.

Figure 7. Critical path in KVIA

### 4. Performance evaluations

In this Section, we evaluate the performance of our KVIA, and compare it with Berkeley-VIA. Table 1 shows our environments for the evaluation. To evaluate the performance, we choose two micro-benchmarks; round-trip latency and one-way bandwidth, and Figure 8 shows each evaluation methodology.

<table>
<thead>
<tr>
<th>Host</th>
<th>processor</th>
<th>Intel Pentium-III 500MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>memory</td>
<td>SDRAM 256Mbytes</td>
</tr>
<tr>
<td>I/O</td>
<td>33MHz 32bit PCI bus</td>
<td></td>
</tr>
<tr>
<td>Myrinet</td>
<td>processor</td>
<td>66MHz 64bit Lanai7.2</td>
</tr>
<tr>
<td></td>
<td>memory</td>
<td>SRAM 4Mbytes</td>
</tr>
<tr>
<td></td>
<td>switch</td>
<td>8-prot switch</td>
</tr>
<tr>
<td>OS</td>
<td>Linux kernel 2.2.13</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Evaluation environments
4.1. One-way bandwidth

First, we report the effect on the parallelization of DMA engines in Myrinet NIC. Figure 9 shows the one-way bandwidth of KVIA. KVIA applied the parallelization of them obtains the maximum one-way bandwidth of 950Mbits per second, which is about 74% of Myrinet link's peak bandwidth, 1.28 Gbits per second. And the half bandwidth is obtained at 4Kbytes data size. As shown two curves in Figure 9, the grower data size is, the more the difference of one-way bandwidth is. This is because the ratio of data transfer latency to overall latency is grower as data size grows. (see Figure 10) In case of transferring small sized data, the spent time for transferring data is less than other wasted time (e.g., protocol processing time of MCP, waiting time for finishing DMA execution), but in case of large sized data, the ratio of them has reversed itself. After all, the improvement of one-way bandwidth is made by automatically hiding the latency for $TxData(RxData)$ or $Transfer(Receive)$, shown in Figure 10, through the parallelization of DMA engines in Myrinet NIC.

Figure 11 shows the one-way bandwidth that is measured when small sized data, less than 256 bytes, is transferred using our adaptive transfer mechanism. At this measurement, the size of the send or receive descriptor is fixed to 48bytes. As shown in Figure 11, KVIA utilized adaptive transfer mechanism obtains about 84Mbits per second bandwidth in transferring 128bytes sized data. Also, overall measured bandwidth is 2.7 to 4.5 times that of only transferring data through DMA engine in Myrinet NIC.

As shown Table 2, the comparison of throughput between PIO write and DMA, while DMA has low transfer rate for small sized data due to start-up latency of DMA engine, PIO is well utilized for them. In addition, VIA protocol requires three movements to send or receive data; descriptor, data and status movement, and even one DMA requires DMA block creation, virtual-to-physical address translation, and so on. However, PIO for small sized data only requires the location information extracted from PIO doorbell. Thus, PIO, used to transfer small sized data, is more applicable than DMA in the utilization of Myrinet link.

<table>
<thead>
<tr>
<th>Size</th>
<th>PIO(Programmed I/O write)</th>
<th>Myrinet DMA Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 bytes</td>
<td>25.46 Mbytes per second</td>
<td>2.55</td>
</tr>
<tr>
<td>8 bytes</td>
<td>30.73</td>
<td>5.06</td>
</tr>
<tr>
<td>16 bytes</td>
<td>29.73</td>
<td>9.68</td>
</tr>
<tr>
<td>32 bytes</td>
<td>30.57</td>
<td>19.70</td>
</tr>
<tr>
<td>64 bytes</td>
<td>31.16</td>
<td>31.34</td>
</tr>
<tr>
<td>128 bytes</td>
<td>34.05</td>
<td>59.74</td>
</tr>
</tbody>
</table>

Table 2. Throughput comparison between PIO write and DMA

4.2. Round-trip latency

Figure 12 shows the comparison of round-trip latency between KVIA and Berkeley-VIA when DMA engines are always used to transfer data without regard to the size of descriptor and data. Raw latency represents the round-trip latency which is obtained as assumed that all data are already located in Myrinet SRAM. As Figure 12 shows,
KVIA provides about 59 microseconds round-trip latency, which is 0.5 ~ 1% improved result comparing with Berkeley-VIA over the same network interface hardware. This improvement is caused by the short doorbell size and the reduction of polling Myrinet SRAM in KVIA MCP. However, two curves of KVIA and Berkeley-VIA represents highly increasing round-trip latency over the *Raw latency* curve, which means that three DMA operations- descriptor, data and status DMA operation- are very expensive in sending or receiving data on KVIA critical path, specially as small sized data.

Figure 12. Round-trip latency measured from KVIA and Berkeley-VIA

Figure 13 shows the round-trip latency of KVIA as applied our adaptive transfer mechanism. As shown Figure 13, for 4 bytes sized data, we obtains about 40 microseconds latency, which is approximately 32% improvement comparing with KVIA which always only uses DMA engines in Myrinet NIC to transfer data. Such a improvement continues up to 256 bytes sized data. The reason about this result can be found in following Tables. The values of each Table exhibit the latency of each item on KVIA’s critical path when the size of send or receive descriptor is fixed to 48 bytes. In Table 3, the values of *TxData* and *RxData* represent the latency of data and status transmission. As shown in Table 3-4, in case of transferring small sized data and descriptor, PIO write to the mapped buffer area is more applicable than DMA in latency. Our adaptive transfer mechanism requires the memory copy of descriptor, data and status in case of transferring data less than 64 bytes. However, it does not bring much latency because the copy size is limited to 64 bytes. (see Table 5) In addition, as shown in Table 3, PIO read from the mapped buffer area takes more latency than PIO write. However, our adaptive transfer mechanism can induce the gain of round-trip latency up to maximum 256 bytes because it adaptively selects applicable transfer mechanism between PIO read and DMA according to the size of received data from Myrinet link.

<table>
<thead>
<tr>
<th>(microsecond/byte)</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>TxTxDesc</em>(MemCopy+PCIWrite)</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><em>TxData</em>(MemCopy+PCIWrite)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>MemCopy in Myrinet SRAM</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td><em>TxRxDesc</em>(MemCopy+PCIWrite)</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><em>RxData</em>(PCIRead+MemCopy)</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>16</td>
<td>28</td>
</tr>
</tbody>
</table>

Table 3. Split time on KVIA critical path as applied adaptive transfer mechanism

<table>
<thead>
<tr>
<th>(microsecond/bytes)</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>TxTxDesc</em>(DMA)</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td><em>TxData</em>(DMA)</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>MemCopy in Myrinet SRAM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td><em>TxRxDesc</em>(DMA)</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td><em>RxData</em>(DMA)</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 4. Split time on KVIA critical path as only used DMA engines

<table>
<thead>
<tr>
<th>bytes</th>
<th>4</th>
<th>16</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>1024</th>
<th>4096</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (microsec)</td>
<td>0.0</td>
<td>0.3</td>
<td>0.6</td>
<td>2.4</td>
<td>12.03</td>
<td>42.06</td>
<td></td>
</tr>
</tbody>
</table>

Table 5. Copy latency in host memory

5. Conclusion

The cluster system aims to provide high performance and scalability for user processes. However, these advantages of the cluster system are mainly limited by inefficient communication protocols. To resolve this problem, researchers proposed user-level communication. VIA is the industry standard for the user-level communication. Although a VIA implementation exists,
the VIA implementation is highly inefficient not considering the characteristics of VIA itself.

In this paper, we design and implement a high performance VIA making use of two transfer mechanisms with different characteristics and the parallelization of DMA engines. Our experimental results shows the round-trip latency of 40 micro-seconds and the maximum one-way bandwidth of 950 Mbits per second, which is about 74% of Myrinet link’s bandwidth. Our VIA implementation can be better applied to the systems that frequently use small messages.

6. References