Human-centric Computing and Information Sciences

RESEARCH Open Access

Compatibility enhancement and performance measurement for socket interface with PCIe interconnections

Cheol Shim, Rupali Shinde and Min Choi*

*Correspondence: mchoi@cbnu.ac.kr
Department of Information and Communication Engineering, Chungbuk National University, Cheongju 28644, South Korea

Abstract
Today the key technology of high-performance computing systems is the emergence of interconnect technology that makes multiple computers into one computer cluster. This technique is a general method in which each constituent node processes its own operation and communicates with different nodes. Therefore, a high-performance network has been required. InfiniBand and Gigabit Ethernet technologies are typical examples of high-performance network. As an alternative to those technologies the development of interconnection technology using PCI Express (Peripheral Component Interconnect Express), officially abbreviated as PCIe or PCI-e, is a high-speed serial computer expansion bus standard, which has characteristics of high speed, low power, and high protocol efficiency is actively under development. In a high-performance network, the TCP/IP protocol consumes CPU resources and memory bandwidth by nature, which is a bottleneck. In this paper, we implement the PCIe based interconnection network system with low latency, low power, RDMA, and other characteristics. We utilize Socket API which is mainly used in the user-level application program rather than the existing MPI and PGAS model interfaces. The implemented PCIe interconnection network system was measured with the metrics as the bandwidth using the Iperf Benchmark which uses the Socket API. The bandwidth at 4 Mbyte of transmission data size was measured to be 1084 Mbyte/s bandwidth based on PCIe, which is about 96 times higher than that of 11.2 Mbyte/s bandwidth based on Ethernet.

Keywords: Cloud computing, Interconnection network, PCIe

Introduction

Semiconductor integration technology have been steadily developed over the last several decades and recently modified into a multicore structure due to problems such as heat generation. As a result, multicore technologies of high-performance computing systems are emerging as systems interconnect technology, which is a technology for making multiple computers into one computer cluster. High-performance computing systems (Computer Clusters) using interconnect technology are essential where artificial intelligence, huge data, and SNS are required. Generally in this technique each constituent node has been processes by its own operation and communicates with different nodes [1]. Therefore, a high-speed computing systems are usually uses a high-speed local area network. In this case InfiniBand and Gigabit Ethernet are typical communication

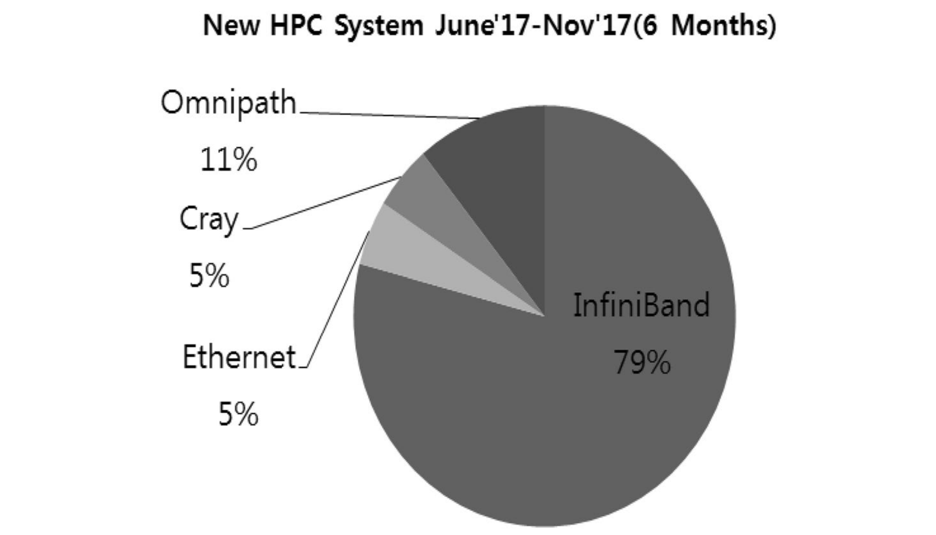


Fig. 1 Percentage of interconnection-based high-performance computing. System developed in June–November

Table 1 Comparison of efficiency of PCIe, Ethernet, and InfiniBand

	PCIe	Ethernet	InfiniBand
Protocol efficiency	93%	86%	88%
End-to-end latency	~1 us	~10 us to 30 us	~1 us to 2 us

networks. Figure 1 shows the percentage of new interconnection-based high-performance computing systems in the 6-month period ending in 2017 [2, 3].

In addition to the above-mentioned InfiniBand and Gigabit Ethernet used in high-performance computing systems, studies are underway to develop a PCIe-based interconnection system using the advantages of PCIe. PCIe is widely used as a standard I/O interface for connecting processors and I/O system devices. High speed, low power, and high efficiency are the salient properties of the PCIe; because of additional properties PCIe is considered as good alternatives to the existing network structures. InfiniBand and Ethernet networks handle TCP/IP packets that are handled in hardware by a Network Interface Card (NIC) device. The network using PCIe without such processing is relatively inexpensive. Table 1 show that the protocol efficiency of PCIeis 5% higher than the InfiniBand, and 7% higher than the Ethernet network. The end-to-end latency is 10 to 30 times less than Ethernet and 2 times less than InfiniBand. It can be seen that the price of the interconnection network is cheaper than the port of the interconnection network [4].

Recently, there are several representative companies that study PCIe related technology, Intel, Dolphinics, and IDT. Dolphinics leverages PCI Express's performance advantages to provide a solution for creating local networks. Utilizing the advantages of PCI Express high throughput and low latency, it enables fast data transfer of storage files and data, and realizes system offload. Dolphinics sells the IXS600 Gen3 switch, a PCI Express switch, and the PXH812 PCI Express Gen3 Host and Target Adapter, a PCI Express

Adpater cards. PCI Express offers low latency and highly efficient switching for high performance applications. Dolphin has implemented a high speed inter-system switching solution using PCI Express technology. The IXS600 PCI Express switch provides a powerful, flexible, and Gen2 switching solution. It utilizes IDT's transparent and NTB (non-transparent bridging) technology to integrate Dolphin's software technology to provide clustering through I/O scaling and inter-processor communication technology. With the IXS600, you can build high-performance computing clusters through multiple PCI Express devices. The IXS600 is a switching device in Dolphin's IX product line, offering 8-port, 1 U cluster switch with ultra-low latency at 40 Gbps of non-blocking bandwidth per port. Each $\times 8$ PCI Express port provides backward compatibility for Gen1 I/O while providing maximum bandwidth per device. The IXS600 switch can be copper or fiber-optic and uses standard iPass connectors [5, 6].

IDT has an extensive product portfolio for building PCI Express networks (switch, bridge, signal integrity, timing solutions, etc.). They provide signal integrity products, e.g., Retimer, Repeater. They also provide switch devices- A device that supports up to 64 lanes, 24 ports, a free port configuration, and a multi-root application based on up to 8 NTB functions, e.g., switch for I/O expansion, switch for system interconnect. In addition, they provide bridge devices, for example, PCIe to PCI/PCI-X Bridge, PCI-X to PCI-X Bridge, PCI to PCI bridge, timing related components such as clock synthesizer, spread spectrum clock generator, PLL zero-delay buffer, jitter attenuators [7].

In this paper, to implement PCIe based interconnection network system with low latency, low power, and RDMA characteristics, we use Socket API, which is mainly used in user-level application of each node which provides a way to utilize the existing Socket application program while providing higher bandwidth in Ethernet based Socket communication for packet transmission through PCIe Switch device driver and Linux Kernel patch. In "Design of enhancing compatibility for socket" section, we describe the design and implementation of the system presented in this paper.

Related work

PCIe was developed to replace the PCI parallel bus, and PCIe uses a bus topology to enable communication between other devices on the bus [8]. It supports multiple lanes of $\times 1$, $\times 2$, $\times 4$, $\times 8$, $\times 16$, and $\times 32$ per link. Data rates are 2 Gbps per lane in PCIe Gen 1, 4 Gbps per lane in PCIe Gen 2 and 8 Gbps in Gen3, and the bandwidth is 128 Gbps and the clock speed is 8 GHz based on the PCIe Gen3 $\times 16$ lane [9].

A PCIe Switch can be a collection of logically connected PCI–PCI bridges. After connecting the additional PCI–PCI bridges downstream, one PCI–PCI bridge is upstream. Thus, PCIe Switching appears as a hierarchical structure of logical PCI–PCI bridges [10]. Figure 2 shows the hierarchical structure provided by one upstream node and the downstream node PCIe switch.

In Multi-Host mode, PEX8749 can be configured with up to six upstream host ports, each with its own dedicated downstream ports. The device can be configured for $1+1$ redundancy or $N+1$ redundancy. The PEX8749 allows the hosts to communicate their status to each other via special door-bell registers [11]. In failover mode, if a host fails, the host designated for failover will disable the upstream port attached to the failing host and program the downstream ports of that host to its own domain. Figure 3, shows the

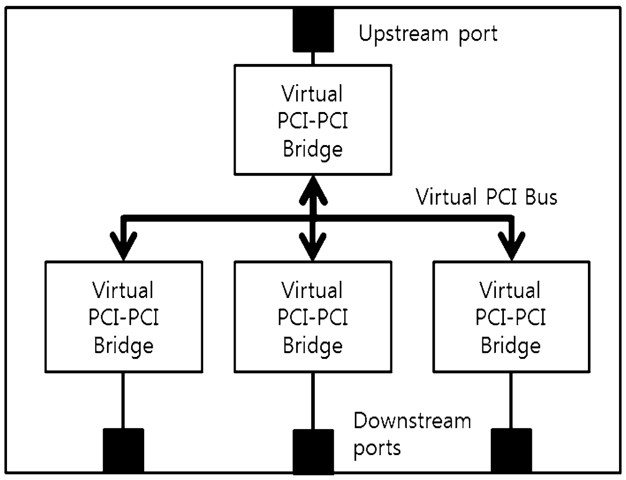


Fig. 2 PCI bridge hierarchy of PCI switch

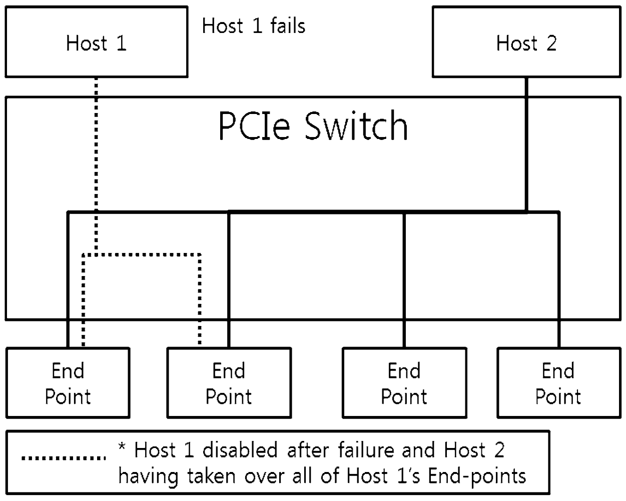

Host 1
Host 1 fails
Host 2
PCIe Switch
End Point
End Point
End Point
End Point
* Host 1 disabled after failure and Host 2 having taken over all of Host 1's End-points

Fig. 3 Fail over operation in PCIe switch multi-host environment [12]

hierarchy when a host fails in a situation where each host has two end-points in multi-host mode. The dotted line is the link to the existing End-Point that Host 1 had. As described above, Host 2 is replaced with the upstream to the endpoints of the existing Host 1 in which the failure occurred.

When constructing an interconnect network using a PCIe switch, communication between each node in a computer cluster uses the PCIe protocol, so that data is not encoded or decoded in a multi-layer protocol. Therefore, unnecessary protocol processing in a network such as Ethernet can be reduced [13]. Also, NTB (Non-Transparent Bridge) technology enables inter-node communication between different PCI domains. The PCIe standard was originally developed for single-host environments. However, as a multi-host network becomes necessary, NTB is the solution to control communication with other nodes in the PCIe network. Several nodes in the PCIe switch can distinguish

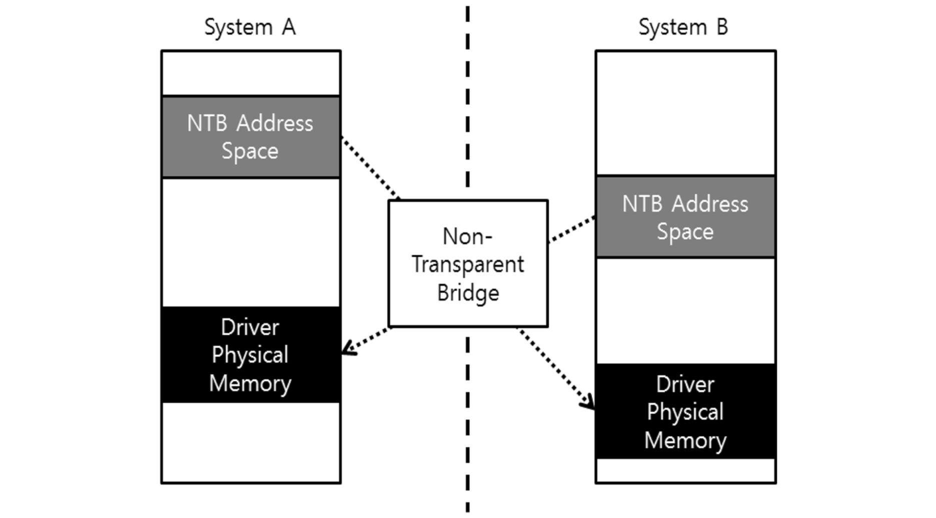


Fig. 4 NTB's remote memory address translation

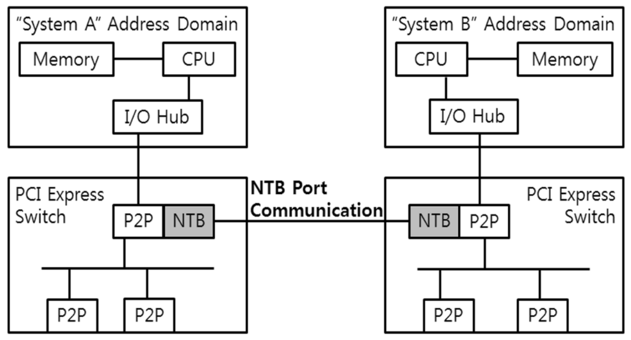


Fig. 5 Communication across NTB ports between different address domains

each node using the Offset of the Base Address Register (BAR) [14]. In addition, due to the characteristics of using SSC (Spread Spectrum Clock), there is a problem that the clock timing conflict and the bus address system conflict. The NTB port logically isolates each node to isolate each system. This problem is solved through the conversion of timing and bus addressing schemes [4]. The nodes of each NTB port are related to some memory area in their memory, so other remote nodes can directly access (RDMA) through address translation [15]. Figure 4 shows the address translation through the NTB port between different nodes.

This NTB port feature has the effect of isolating two different systems on the PCIe bus [16, 17]. However, since the above-mentioned RDMA is possible, it is also possible to communicate with each other through address translation. Figure 5 shows the communication between NTB ports between systems with different address domains.

The application performance of a computer cluster depends on the network performance of the LAN or SAN connecting each node. Generally, in a SAN (System Area Network) environment where clusters are configured, it is safe against data loss during data transmission and reception. Therefore, functions such as error checking and flow control provided by the TCP/IP protocol act as an overhead. Several communication protocols such as FM (Fast Message), U-Net, and VMMC (Virtual Memory-Mapped Communication) have been proposed to solve TCP/IP problems in a SAN environment [15]. Based on these protocols, VIA (Virtual Interface Architecture) [18] has been proposed for low latency and high bandwidth networks. InfiniBand, RDMA over Converged Ethernet, iWARP RDMA Protocol) exists.

A technology commonly used in high-performance networks is Remote Direct Memory Access (RDMA). Since the CPU directly accesses the memory of the remote node to transmit and receive data, the CPU overhead due to the protocol processing can be reduced and the communication performance can be maintained.

VIA is a network abstraction model that supports InfiniBand, RoCE, iWARP technology, and supports Zero-Copy to minimize RDMA support and buffer-to-buffer copy [19]. In addition, VIA communicates by writing and reading data directly to the memory area between each node, unlike the existing network where the protocol stack operates as software in the Kernel domain.

High-performance interconnect technologies using RDMA use drivers, RDMA operations, User-level API, and MPI provided by Open Fabrics Enterprise Distribution (OFED) middleware. To use RDMA APIs such as connectivity, parallel processing, and network control in applications, we use functions called Verbs. IP over InfiniBand (IPoIB), which is widely used among the above-mentioned high-performance interconnection, provides a function to use existing Ethernet-based applications without major source code modification [20]. The OFED API is available on a variety of operating systems, including Red Hat Linux, Oracle Linux, and Windows Server.

Traditional TCP/IP-based Socket communication is not suitable for high-performance computing systems such as InfiniBand. TCP/IP Sockets depend on the kernel to send and receive messages, which increases the latency time by the frequent occurrence of Kernel Context Switches. The TCP/IP Socket method copies data from the sending application to the kernel buffer and sends it to the NIC kernel buffer. The receiving application copies data to the kernel buffer into the NIC buffer and then copies the data back to the application's buffer (Fig. 6).

The TCP/IP communication method consumes CPU resources and memory bandwidth as the network speed increases in the process of data recombination and transmission, which causes bottleneck [22]. Nevertheless, many existing applications

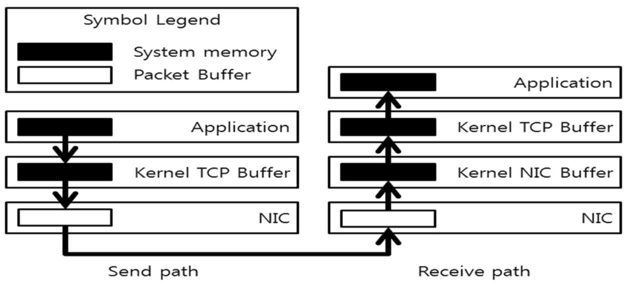


Fig. 6 Process of sending/receiving TCP/IP data through NIC [21]

communicate with each other through Socket, and SDP (Socket Direct Protocol) is used to solve this problem [23, 24].

Single-copy, a technique to prevent unnecessary copying, has been proposed to optimize TCP/IP [25]. However, if the physical speed of the network exceeds the gigabit level, there is a limit to overcome the overhead caused by the CPU handling TCP/IP. In recent years, InfiniBand and Ethernet NICs have improved performance by adding a TCP Offload Engine (TOE) function to overcome the drawbacks of TCP/IP. The TOE can reduce the cost of the protocol processing by the CPU by directly processing the TCP/IP packets processed by the operating system in the NIC. In addition, since the NIC handles communication, the communication performance can be maintained even when the CPU load increases [26].

Figure 7 is a flowchart of data communication between application programs by a ZCopy method in InfiniBand interconnection system. First, a buffer is allocated to send data of the receiving application program. If the buffer is large enough, send the SrcAvail message to the sender and register the source of the data. The buffer is then sent to the receiver and the RDMA Read is started to read the data from the receiving application. When RDMA ends, the RDMARdCompl signal is sent to the transmitter and the communication is terminated [27].

In "Design of enhancing compatibility for socket" section of this paper, we propose a PCIe interconnection-based RDMA with low latency, high protocol efficiency, and low cost, which reduces Socket communication overhead in interconnection networks based on TCP/IP and SDP protocols such as InfiniBand and Gigabit Ethernet. The system designed and implemented in the Socket communication system using the communication system will be described.

Design of enhancing compatibility for socket

This section explains device driver implementation and kernel level patch for implementing PCIe interconnection network system using a Socket interface as mentioned above. The system to be implemented in this paper aims to operate on Broadcom's PLX PCIe Switch PEX-8749 and PLX PCIe NIC PLX-8749.

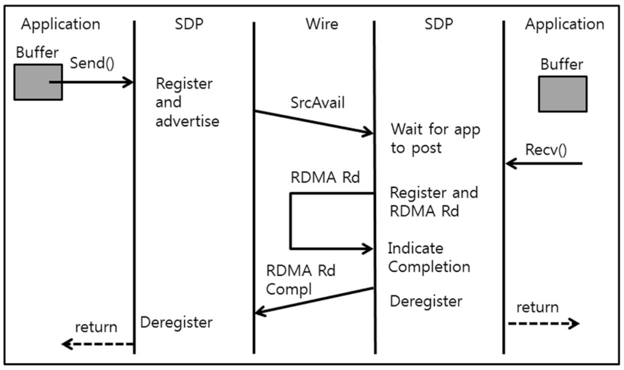


Fig. 7 ZCopy read communication processing flow chart

Overall architecture

When configuring a PCIe interconnection network, each node will send and receive data through the PLX PCIe Switch PEX-8749. When the application program of each node reads/writes for Socket communication, it DMAs to the memory area of another node through the BAR provided by the NTB port and communicates.

Interconnect devices such as InfiniBand and Ethernet are supported by a user-level library or device driver that requests a DMA bus address as a relative node, but PCIe switches do not have the hardware capability to support Socket communication. Therefore, when sending and receiving through Socket, it is necessary to obtain the DMA bus address for the other node. In the PCIe Switch, the NTB port has a Scratchpad register that can be shared and accessed by two different systems, and a Doorbell register that can generate an interrupt between logically isolated systems due to the NTB. It also maps the actual physical address area to the memory area allocated by the application program. This memory area is used as a memory for Socket communication. When an application calls Socket API, it reads the memory of the other node and sends and receives data to and from each other. Figure 8 shows the process of accessing each other's address area after getting through the NTB port provided by PCIe Switch.

In general Socket communication, the application program creates a Socket through a system call at the User level and proceeds to the actual read/write using the file descriptor for this Socket in the kernel area.

The Socket communication method based on the PCIe switch proposed in this paper treats Socket APIs in the application program only for the Socket which needs DMA transmission in the read/write function called in the kernel area. In the read/write function in the kernel, communication is performed using a DMA device by branching to a specific port. If the application creates a Socket and binds the specified port number for Socket communication via PCIe Switch, I/O for TCP/IP through the File Descriptor for the device file for I/O in the kernel area, I/O for DMA can be confirmed. Figure 9 shows how to determine the communication method according to the branch inside the Write function in the kernel area when calling the Socket API in this application program. In this case, the application program uses the Socket interface without changing a lot of source code, we expect to improve performance.

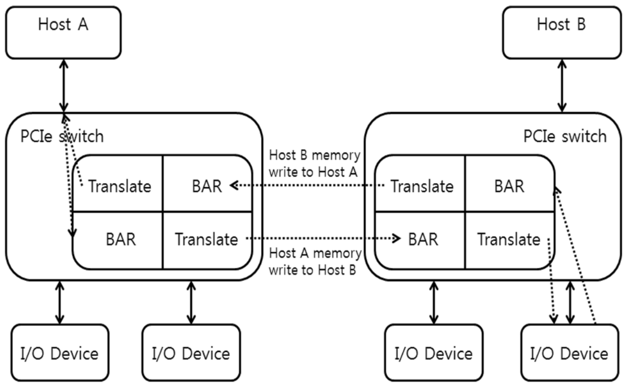


Fig. 8 Host A and B via NTB access each other's memory area

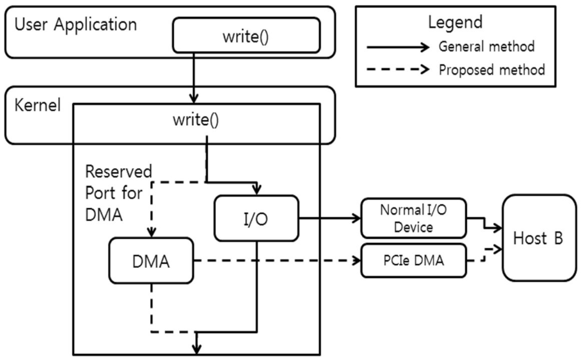


Fig. 9 Socket branch when DMA is required in write function of Kernel area

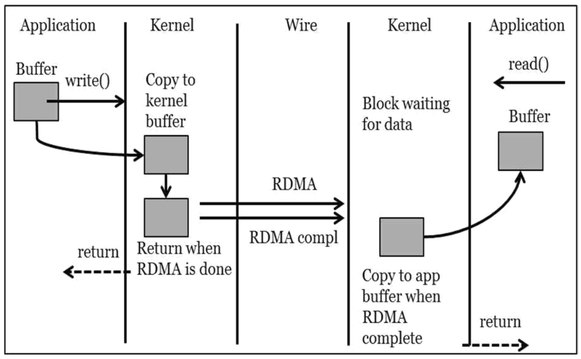


Fig. 10 Socket program flow diagram with PCIe switch

Figure 10 shows the flow of communication in the previously designed system. The sender and receiver application program prepares a buffer for data transmission and reception, which maps to the BAR register for RDMA to the peer obtained via the NTB port. As mentioned in "Related work" section, we use the Scratchpad register of the NTB port to map a certain part of the memory area of our own to the memory area of the other node and map them to each other. When the data is transmitted through the RDMA engine to the specific memory area mapped to the memory of the correspondent node which can receive the data, the data is also transmitted to the real memory area of the correspondent node. In order to signal the end of the data transfer, it is necessary to cause an interrupt to the partner node. To do so, a Doorbell register is provided which can generate an interrupt at the partner node. If a bit that causes an interrupt to the partner node is set in its Doorbell register to indicate the end of data transmission at the node receiving the data, the Doorbell interrupt is generated at the counterpart node so that the end of the data transfer can be recognized.

When the application program of the transmitting node calls the write() function for Socket communication, the data buffer is copied to the kernel area. The receiving node's application program calls the data read() function to wait for the RDMA to be written into the memory area mapped to the BAR register of its NTB port from the transmitting node. And RDMA writes the data to be sent to the memory area for the

RDMA of the receiving node obtained from the transmitting node through the NTB port. Then check whether the DMA is working properly in the kernel write() function. When the DMA transfer is completed, the send node uses the Doorbell register of the receiving node NTB port to terminate the write() function while generating an interrupt. This is to prevent the receiving node from reading the data in its RDMA memory area until the Doorbell interrupt occurs on the receiving node side. The receiving node that received the Doorbell interrupt that the RDMA data transmission is completed has the buffer for data reception of the application program mapped to the memory area for RDMA as well. Therefore, the data of this area is copied from the Kernel area to the data buffer of the application program.

Socket interface access implementation on kernel level

The socket is created in the application program, and the read/write function of the kernel area is actually called through the Socket API read/write interface used at the user level. In the function of this kernel area, RDMA should be performed for the port designated for Socket communication based on PCIe Switch as designed above. When calling the read() function and the write() function in the application program in the Socket communication, the file descriptor of the socket is brought as a parameter in the read() function and the write() function of the kernel area. In order to confirm that the File Descriptor is a Socket type file descriptor, it is checked whether it is registered in the Socket look-up table. If it is a file descriptor created with Socket, as shown in Fig. 9 to read the port number bound in the inet_sk macro supported by Linux and determine whether to perform DMA communication. Some modifications to the Linux kernel source code fs/read_write.c to be implemented for these kernel patches are required.

Since the DMA communication is performed through the PLX PCIe Switch, the device file of the PEX-8749 device is read in order to call the DMA communication function registered in the file operation of the PLX SDK device driver module, and the FILE structure is obtained from this device file. The obtained FILE structure can access the functions belonging to the File Operation of the PLX SDK device driver. This file operation can call the read() function and the write() function implemented for this system in the PLX SDK device driver modified to use the RDMA function of the PLX PCIe Switch described in the next section. When the Socket application calls the Socket API read() and write() APIs, the function that performs the RDMA function of the PLX PCIe Switch is called in the kernel area.

To do this, you must first register the address of the PLX PCIe DMA device driver in the FILE pointer before starting Socket communication. Figure 11 shows the process of acquiring the address of the PLX PCIe device driver. Before starting communication, open the PLX PCIe DMA device file through the sys_open() function in the kernel area. Get the file descriptor with fdget() function using the number of the file. This address is the address of the PLX PCIe Switch DMA device driver. If you are using Socket communication via PCIe Switch.

The file pointer obtained from the above process contains the address of the device driver for the DMA device of the PLX PCIe Switch and is still used when performing DMA Socket communication through the PCIe Switch. Figure 12 shows the process of

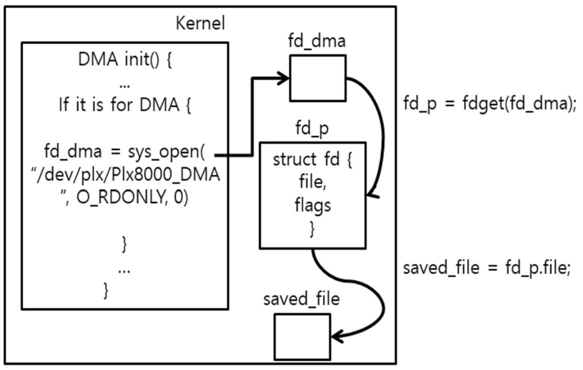


Fig. 11 Procedure for obtaining the file address of the PLX PCIe DMA device driver

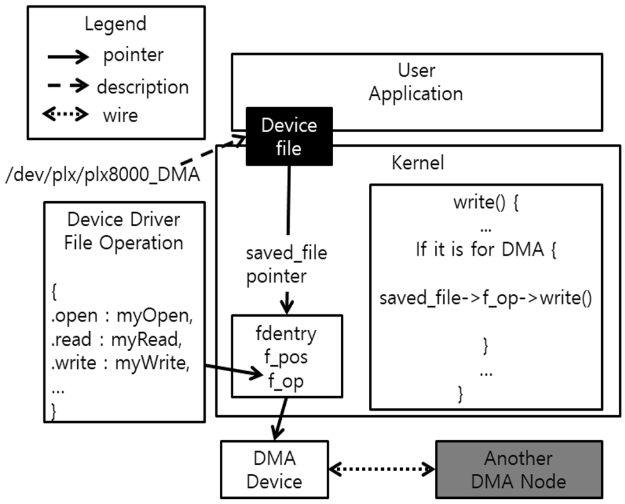


Fig. 12 Procedure for accessing registered PLX device file and file operation

calling the function of File Operation with saved_file->f_op-> write() using this saved file pointer in the kernel area.

Socket interface access implementation on device driver

To use the PCIe Switch PEX-8749, PEX-8732 Adapter, and PCIe PLX-8749 NIC, PLX provides a PLX SDK device. It is necessary to map the BAR (Base Address Register) of the NTB port of the PCIe Switch and the physical memory of the node itself and to map the data buffer so that the application can access this memory area. This mapped memory is used as space for RDMA transmission.

When an application program calls the read() and write() APIs for Socket communication using DMA, it is mapped to its own memory area using the DMA read/write function registered in the File Operation of the PLX SDK device driver DMA can be performed in the memory area of the other party.

First, when the PLX SDK device driver is inserted into the kernel, the initialization function of the device driver provides an a_init function with an integer return value and no parameters. In this initialization function, a function for DMA read/write is registered and a memory to be used in the PCIe Switch network is allocated. It also allows you to register the functions of this device driver's File Operation, which are the functions of the device driver to be called from within the kernel read() and write().

The implementation of the write() function of the PLX PCIe device driver to use the DMA function is as follows. DMA register information before DMA start and channel offset information to be used by each node are recorded in a register for DMA function. This process supports the PLX_DMA_REG_WRITE macro in the PLX SDK device driver. Here, the information to be included in the DMA register includes a memory area of data necessary for actual transmission, which is input as a parameter of the write() function of the application program. In this memory area, data is transferred from the application program to the kernel area and copied to the memory area of the own node mapped to the real memory area of the partner node through the NTB port. Then, DMA transfer is started by setting the Start bit in the DMA register. Figure 13 shows this process.

The PLX SDK device driver also supports the PLX_DMA_REG_READ macro, which can read the information of the register. By using this macro, the In-progress bit indicating the progress of the channel used for the DMA transfer is confirmed. If cleared, the DMA is terminated, if it is still set, it can be seen that the DMA transfer is in progress.

When implementing the system using the method described above, the DMA engine operates independently of the CPU, so the application program will normally operate to receive and transmit data from read() and write() using the DMA function of the PCIe switch.

However, if the data transfer end state of the DMA channel is not known, it cannot be known whether or not all the data transmitted from the transmitting node to the memory area of the receiving node in the Socket communication is transmitted. This causes a situation in which the receiving node can DMA again before the receiving node receives all the data, and the receiving node overwrites the existing data before reading the existing data in the memory. It is hard to say that Socket communication was performed normally. Figure 14 shows the process of simply calling DMA in the

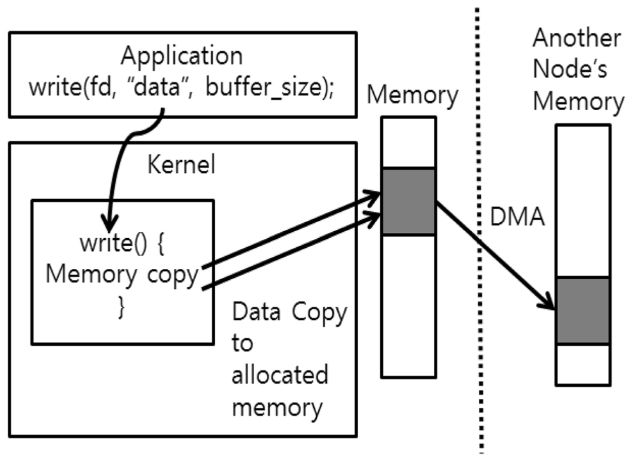


Fig. 13 Copy and DMA process of data required for DMA in Socket communication

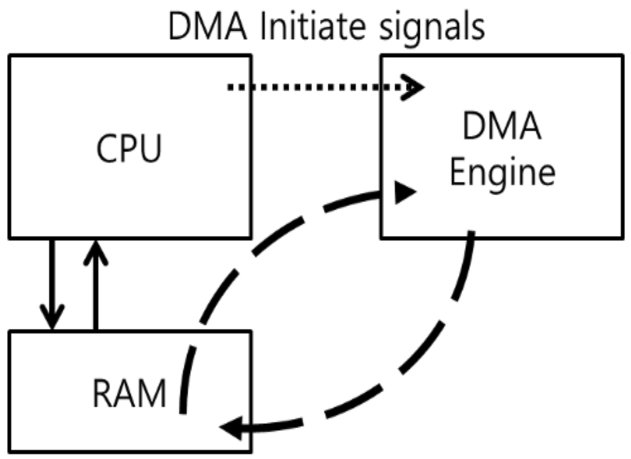


Fig. 14 Measurement issues when performing DMA in existing write()

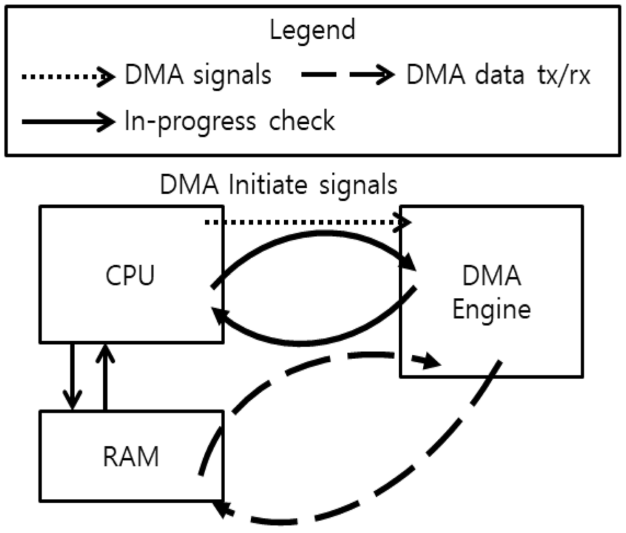
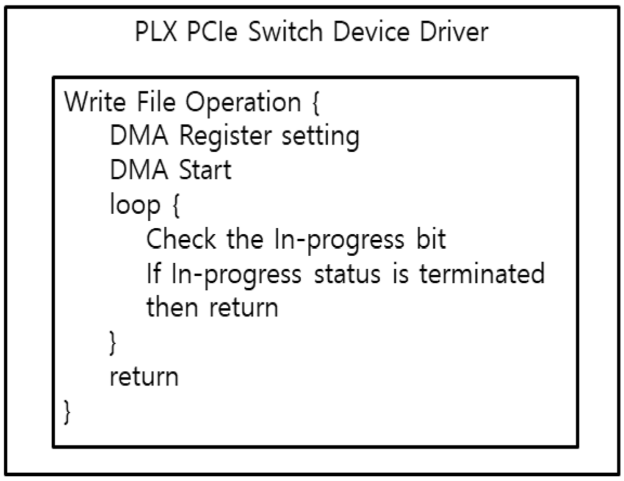


Fig. 15 Improvement plan for checking DMA transfer status

existing write() system call. Since the CPU does not actually participate in the DMA communication, it is difficult to obtain the time required for data transmission, that is, the bandwidth.

To solve this problem, the In-progress bit is used to check the progress of the DMA register of the channel being used by the DMA register. In the application program, the write file operation function of the PLX PCIe Switch device driver, which is called by the write() system call, is implemented in such a manner that the DMA status register in-progress bit is polled to check whether the DMA is terminated. That is, when the DMA transfer to the other node is completed, the write() function is returned to complete the data transfer. Figure 15 shows the improvement method using the in-progress check of this Polling method. Figure 16 shows the source code structure of the Write File Operation described above.



```
Write File Operation {
    DMA Register setting
    DMA Start
    loop {
        Check the In-progress bit
        If In-progress status is terminated
        then return
    }
    return
}
```

Fig. 16 Write file operation source code structure

Table 2 Hardware configurations

Main board	**GIGABYTE GA-H61M-DS2V**
Host PC hardware	
Processor	Intel Core i5-3470 CPU @ 3.20 GHz * 4
Memory	Samsung DDR3 1333 MHz 2 GB
Operating systems	Linux CentOS 7 64 bit Kernel : 3.10.0-327.el7 (patched)
Interconnection network configurations	
PCIe switch	PLX PEX8749 RDK 48 Lane, 18 Port, PCIe PCIeGen 3 Switch (Gen 3)
NIC	PLX PEX8732 Cable Adapter * 4
Device driver	PLX SDK's Reference Device Driver (patched)

When the Socket application calls the Socket read() and write() APIs through the PLX PCIe Switch device driver, the read() and write() functions in the kernel are called and the PLX PCIe Switch's File Operation function to transmit and receive data. We designed and implemented a system for using the Socket interface in a network based on PLX PCIe interconnection. The following section describes the results of comparative analysis of the performance of such systems.

Experimental results

In this paper, Iperf benchmark (version 2.0.5) similar to Netperf was used for performance evaluation of PCIe based Socket communication system designed and implemented in "Design of enhancing compatibility for socket" section. Iperf is an open-source benchmark for TCP, UDP, and SCTP communications using Socket, which measures bandwidth according to data length and size [28].

The PCIe interconnection system environment was implemented using a Broadcom PLX PCIe Switch PEX-8749 and a PLX PCIe NIC PLX-8749 connected to an 8-lane PEX-8732 Cable Adapter. Table 2 shows the hardware configurations of the

Intel I5-3470 processor, 2 GB of memory, patch kernel of CentOS 7 Kernel 3.10.0, and the PLX SDK device driver.

Two hosts configured with the above environment are connected to the NTB port of the PLX PCIe Switch PEX-8749. In order to evaluate the performance of PCIe based Socket communication system and Ethernet based Socket communication, Iperf Benchmark divided into data bandwidth measurement using RDMA and data bandwidth measurement using TCP/IP when calling Socket API. The nodes consisting of two hosts send and receive data using the server-client model as a client node transmitting data and a server node receiving data, respectively.

The data size for transmitting and receiving bandwidth is 1 Byte, 2 Byte, 4 Byte, 8 Byte, 512 Kbyte, 1 Mbyte, 2 Mbyte, and 4 Mbyte, and compared the bandwidth difference between transmission and reception according to each data length. Through the Iperf Benchmark, two hosts act as nodes acting as servers and as clients acting as clients, respectively. The node that is acting as the server continues to receive data from the client node. It waits until the In-progress bit of the DMA status register changes to indicate that RDMA is completed in the client node according to the data transfer. When the application prepares the data buffer and confirms the RDMA completion status, it can calculate the bandwidth using the time difference from the client node to the in-progress bit by polling method and the time until the write() function is returned.

Figure 17 and Table 3 show the results of the bandwidths measured according to the PCIe-based Socket communication and the Ethernet-based TCP/IP Socket communication, Graph.

As a result of analysis based on Table 3 and Fig. 17, the bandwidth increases until the data size sent to the Socket communication becomes larger than the maximum size of the DMA buffer. This can provide enough bandwidth to increase the amount of data to be sent when the DMA buffer size is sufficient, but if the amount of data to be sent reaches the maximum size of the DMA buffer, there will be no buffer space.

In the case of 4 Mbyte of data transmitted through Socket, the bandwidth of PCIe-based Socket communication system proposed in this paper is 1084 Mbyte/s, which is about 96 times higher than 11.2 Mbyte/s bandwidth of Ethernet based Socket communication system respectively.

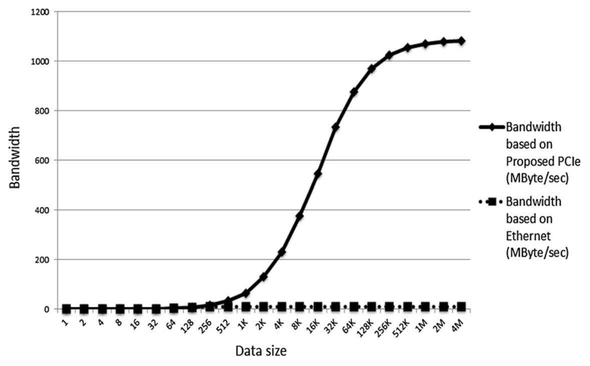


Fig. 17 Differences in PCIe and Ethernet bandwidth depending on data size

Table 3 DMA and Ethernet bandwidth results according to the data size

	DMA Bandwidth (Mbyte/s)	TCP/IP Bandwidth (Mbyte/s)
1	0.06	0.07
2	0.13	0.13
4	0.26	0.26
8	0.51	0.53
16	1.02	1.06
32	2.09	2.11
64	4.12	4.23
128	8.21	8.46
256	16.4	11.2
512	32.9	11.2
1K	64.7	11.2
2K	132	11.2
4K	230	11.2
8K	376	11.2
16K	547	11.2
32K	734	11.2
64K	876	11.2
128K	971	11.2
256K	1025	11.2
512K	1054	11.2
1M	1071	11.2
2M	1080	11.2
4M	1084	11.2

Conclusion

In this paper, we propose a PCIe-based interface with high-speed, low-power, high protocol efficiency using Socket interface instead of MPI standard or PGAS programming model used in existing high-performance interconnection systems such as InfiniBand and Gigabit Ethernet. We implemented a connection network system. When an application calls the Socket interface in a PCIe Switch network that enables PCIe interconnection network configuration, RDMA through address switching for each node using NTB port is used for communication instead of the existing protocol.

In the implemented PCIe interconnection system, the system for utilizing the Socket interface measures performance through Iperf Benchmark open source benchmark tool which measures the performance by Socket communication according to the length of data with protocols such as TCP/IP, UDP, and SCTP respectively. The PCIe switch was used to the Broadcom PLX PCIe PEX-8749 and the PLX PCIe NIC PLX-8749, which were connected to the PEX-8732 Cable Adapter 8 lane. The experimental results compared with the bandwidth of Socket communication based on PCIe interconnection and Socket communication based on Ethernet. The data size used for Socket write() 1 Byte, 2 Byte, 4 Byte, 8 Byte,…, 512 Kbytes, 1 Mbyte, 2 Mbyte, and 4 Mbyte.

Although the bandwidth of the PCIe-based Socket communication and the Ethernet-based Socket communication did not greatly differ between 1 byte and 128 bytes, the bandwidth of 4 Mbyte was 11.2 Mbyte/s and PCIe-based, the performance is about 96 times higher than the bandwidth of 1084 Mbyte/s.

In the future research plan, the polling method is not used in order to reduce the overhead due to the DMA end status. To check the data communication synchronization in the transmitting node designed in this paper, and the ending state of the data transmission is informed to the transmitting node at the receiving node, we can expect higher performance if we optimize our system. These device driver level patches will be improved in the utilization and performance of PCIe interconnection system using Socket interface.

Authors' contributions
Cheol Shim, Shinde Rupali and Min Choi conducted a comprehensive research of socket interface performance measurement and compatibility enhancement on PCIe bus based interconnection network. All authors read and approved the final manuscript.

Acknowledgements
This work has been performed strategic research project of NRF-2017R1E1A1A01075128 by National Research Foundation.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

Funding
This research was funded by National Research Foundation (Grant number : NRF-2017R1E1A1A01075128).

Publisher's Note

Received: 10 October 2018 Accepted: 25 February 2019
Published online: 12 March 2019

References
1. Cho K, Kim J, Dduan D, Jeong W, Park J, Lee Y, Kim H, Jeong J, Shin J, Lee J (2016) Big data analysis based HPC technology trends using Supercomputers. Korea Inst Sci Eng Mag 34(2):31–42
2. The Next platform https://www.nextplatform.com/2017/11/13/top500-dead-long-live-top500/
3. Expósito RR (2013) Performance analysis of HPC applications in the cloud. Future Gen Comput Syst 29(1):218–229
4. Kim Y, Learn Y, Choi W (2015) Design and implementation of PCI express based system interconnection. J Inst Elect Inf Eng 52(8):74–85
5. Zhang L, Hou R, McKee SA, Dong J, Zhang L (2016) P-Socket: optimizing a communication library for a PCIe-based intra-rack interconnect
6. Ullah F, Abdullah AH, Kaiwartya O, Kumar S, Arshad MM (2017) Medium access control (MAC) for Wireless body area network (WBAN): superframe structure, multiple access technique, taxonomy, and challenges. Hum Comput Inf Sci 7(1):34
7. Choi M, Park JH (2017) Feasibility and performance analysis of RDMA transfer through PCI express. J Inf Process Syst 13(1):95–103
8. Ravindran M (2007) Cabled PCI express-a standard high-speed instrument interconnect. In 2007 IEEE autotestcon
9. Meduri V (2011) A case for PCI express as a high-performance cluster interconnect. HPC Wire 28:33
10. Mayhew D, Venkata K (2003) PCI express and advanced switching: evolutionary path to building next generation interconnects. In: 2003 proceedings of 11th symposium on IEEE
11. Ahmed BK (2012) Socket direct protocol over PCI express interconnect: design, implementation and evaluation. M.S. thesis, Simon Fraser University
12. Broadcom. PLX PCI Express switch PEX8749 product brief https://docs.broadcom.com/docs/12351856?eula=true
13. Choi W, Kim Y, Bae S, Kim W (2015) Design of specialized communication module for PCI express network devices. In: Proceeding of Korean Institute of Communications and Information Sciences
14. Onufryk PZ, Tom R (2008) Expansion of cross-domain addressing for PCI-express packets passing through non-transparent bridge. US Patent No. 7,334,071. 19 Feb. 2008
15. Jung IH, Chung SH, Park S (2004) A VIA-based RDMA mechanism for high performance PC cluster systems. J KIISE 31(11):635–642
16. Dutta S, Murthy AR, Kim D, Samui P (2017) Prediction of compressive strength of self-compacting concrete using intelligent computational modeling. Comput Mater Contin 53(2):157–174

17. Koo Kyungmo, Junglok Yu, Kim Sangwan, Choi Min, Cha Kwangho (2018) Implementation of multipurpose PCI express adapter cards with on-board optical module. J Inf Process Syst 14(1):270–279
18. Dunning D (1998) The virtual interface architecture. IEEE Micro 18(2):66–76
19. Choi S, Moon Y, Choi M (2017) RDMA based high performance network technology trends. J Korean Inst Commun Inf Sci 42(11):2122–2134
20. Kashyap V (2006) IP over InfiniBand (IPoIB) architecture. http://buildbot.tools.ietf.org/html/rfc4392
21. Dell. TCP Offload Engines http://www.dell.com/downloads/global/power/1q04-her.pdf
22. Goldenberg D (2005) Transparently achieving superior socket performance using zero copy socket direct protocol over 20 Gb/s InfiniBand links. In: 2005 IEEE cluster computing
23. Balaji P (2004) Sockets direct protocol over InfiniBand in clusters: Is it beneficial? In: IEEE international symposium on-ISPASS performance analysis of systems and software
24. Balaji P (2006) Asynchronous zero-copy communication for synchronous sockets in the sockets direct protocol (SDP) over InfiniBand. In: 20th international IEEE parallel and distributed processing symposium
25. Camarda P, Pipio F, Piscitelli G (1999) Performance evaluation of TCP/IP protocol implementations in end systems. IEEE Proc Comput Digit Tech 146(1):32–40
26. Jang H, Oh SC, Chung SH, Kim DK (2005) Analysis of TCP/IP protocol for implementing a high-performance hybrid tcp/ip offload engine. J KIISE 32(6):296–305
27. Goldenberg (2005) Zero copy sockets direct protocol over infiniband-preliminary implementation and performance analysis. In 13th symposium on IEEE
28. Iperf. Iperf Benchmark v2.0.5 https://iperf.fr/iperf-download.php