2013 IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum

Tightly Coupled Accelerators Architecture for Minimizing Communication Latency among Accelerators

Toshihiro Hanawa, Yuetsu Kodama, Taisuke Boku, and Mitsuhisa Sato
Center for Computational Sciences
University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305-8577 Japan
Email: hanawa@ccs.tsukuba.ac.jp, {kodama,taisuke,msato}@cs.tsukuba.ac.jp

***Abstract*—In recent years, heterogeneous clusters using accelerators have been widely used in high performance computing systems. In such clusters, inter-node communication among accelerators requires several memory copies via CPU memory, and the communication latency causes severe performance degradation. In order to address this problem, we propose the Tightly Coupled Accelerators (TCA) architecture to reduce the communication latency between accelerators over different nodes. In addition, we promote the HA-PACS project at the Center for Computational Sciences, University of Tsukuba, in order to build up the HA-PACS base cluster system, as a commodity GPU cluster, and to develop an experimental system based on the TCA architecture as a proprietary interconnection network connecting accelerators beyond the nodes. In the present paper, we describe the TCA architecture and the design and implementation of PEACH2 for realizing the TCA architecture. We also evaluate the functionality and the basic performance of the PEACH2 chip, and the results demonstrate that the PEACH2 chip has sufficient maximum performance with 93% of the theoretical peak performance and a latency between adjacent nodes of approximately 0.8μsec.**

***Keywords*-GPGPU; Accelerator Computing; Interconnection Network; PCI Express; Remote DMA; CUDA; GPU Direct**

I. Introduction

Accelerators, as represented by Graphics Processing Units (GPUs), are widely used in high performance computing due to their high performance/price ratios and high performance/power ratios. In the Top500 List, the number of high performance computing systems that include accelerators has increased yearly, and several dozen systems use accelerators in the current list (Nov. 2012) [1].

However, as the interface between the CPU and accelerators, PCI Express (PCIe) performance becomes a bottleneck, while the accelerators provide high computing performance. In the cluster using accelerators, inter-node communications among multiple accelerators is required. Conventionally, due to inter-node communication, accelerators must communicate with other accelerators in different nodes through several memory copies via the CPU memory. The latency caused by multiple memory copies severely degrades the performance, especially in the case of a short message.

In order to address this problem, we propose the Tightly Coupled Accelerators (TCA) architecture, which consists of a proprietary interconnect for accelerators, which improves the communication latency among accelerators over different nodes, in addition to the configuration of the commodity cluster using accelerators. The TCA architecture can eliminate protocol overhead, such as that associated with InfiniBand and MPI, as well as the memory copy overhead because TCA enables direct communication via PCI Express. The TCA architecture provides a virtual extension of the PCI Express address domain to a sub-cluster including several nodes, as if an accelerator in a different node existed in the same node. Such capability helps the programmer's view since explicit communication, for example using MPI, is no longer needed at the sub-cluster level similar to a communication among the GPUs in the node.

In the present paper, we describe the TCA architecture and its implementation, namely, PCI Express Adaptive Communication Hub version 2 (PEACH2). Moreover, we confirm the functionality of and investigate the basic performance of the PEACH2 chip through a preliminary evaluation. In Section II, we introduce the HA-PACS project and the HA-PACS system including the base cluster and HA-PACS/TCA at the Center for Computational Sciences, University of Tsukuba. We explain the design of the PEACH2 chip, based on the concept of PCIe direct link, i.e., PCI Express Adaptive and Reliable Link (PEARL) in Section III. The results of the preliminary evaluation using PEACH2 are presented in Section IV. Related research is discussed in Section V, and conclusions are presented and future research is discussed in Section VI.

II. Overview of the HA-PACS System

The Highly Accelerated Parallel Advanced system for Computational Sciences (HA-PACS system) is the eighth generation of the PACS/PAX series supercomputer at the Center for Computational Sciences, University of Tsukuba. In the HA-PACS project, target applications, including particle physics, astrophysics, and life sciences applications, are pre-defined. Within limited power and a limited installation area, a large-scale cluster using accelerators is introduced in order to effectively perform these applications.

978-0-7695-4979-8/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPSW.2013.226

Figure 1. Photograph of the HA-PACS Base Cluster System with 26 racks

The HA-PACS system consists of two parts: the HA-PACS base cluster and HA-PACS/TCA, which is an extension of the HA-PACS base cluster.

A. HA-PACS base cluster

For the development and product-run of advanced scientific computations toward next-generation accelerated computing, the HA-PACS base cluster is equipped with the latest GPUs, as the accelerators, at the time of installation.

Each computation node of HA-PACS uses two sockets of an Intel Xeon E5 2600 (Sandy Bridge-EP) CPU and four NVIDIA M2090 GPUs. Each CPU provides x40 lanes of PCIe Gen.3 directly. Therefore, two CPUs can support four GPUs with full bandwidth of PCIe I/O capability and an additional two x8 lanes can be used for the interconnect NIC and other components.

As an interconnection network, InfiniBand QDR with a dual-rail configuration is used because the NIC uses PCIe Gen3 x8 and the interface can provide approximately 8 Gbytes/sec. A total of 268 computation nodes are connected by the fat-tree configuration with full bisection bandwidth.

Table I shows the specifications of the HA-PACS base cluster. The system began operating in February 2012 with a peak performance of 802 TFlops and was ranked at 41st on the Top500 list issued June 2012. The system exhibited a good performance/power efficiency of 1.04 GFlops/W.

B. HA-PACS/TCA: Tightly Coupled Accelerators

For the accelerator technology of the next-generation HPC, improvement of the latency and bandwidth in the communication among accelerators is the most important issue. In order to address this problem, we propose a novel interconnect dedicated to direct communication among accelerators beyond nodes. This technology is referred to as the TCA architecture.

The TCA architecture is based on the concept of using the PCIe link as the communication network link between communication nodes and GPUs rather than simply using the PCIe link as a within-node I/O interface. In the TCA architecture, the basic unit is the sub-cluster, which consists of eight to 16 nodes. Since the length of the PCIe external cable is limited to several meters and a large number of nodes degrades the performance, it is inefficient to generate a large-scale cluster that includes several hundreds of nodes. Like the base cluster system, each node is also equipped with the InfiniBand interface. Therefore, HA-PACS/TCA can use a hierarchical network that incorporates TCA interconnect for local communication with low latency and InfiniBand for global communication with high bandwidth.

A GPU with Kepler architecture by NVIDIA Corp. is used as the accelerators in HA-PACS/TCA [2]. In this type of GPU, GPUDirect Support for RDMA [3] is available with the CUDA 5 programming environment [4], and we realize TCA communication using this mechanism.

Table I
SPECIFICATIONS OF THE HA-PACS BASE CLUSTER

Computation Node	
CPU	Intel Xeon-E5 2670 2.6 GHz × two sockets (eight cores + 20-Mbyte cache) / socket
Memory	DDR3 1600 MHz × 4 ch, 128 Gbytes
Peak performance	332.8 GFlops
GPU	NVIDIA Tesla M2090 1.3 GHz × 4
Memory	GDDR5 6 Gbytes / GPU
Peak performance	2660 GFlops
InfiniBand	Mellanox Connect-X3 Dual-port QDR
System Specifications	
Number of nodes	268
Storage	Lustre File System 504 Tbytes
Interconnect	InfiniBand QDR 288 ports switch × 2
Total peak performance	802 TFlops
Number of racks	26
Maximum power consumption	408 kW

III. HA-PACS/TCA WITH PEACH2

HA-PACS/TCA is constructed using the PCIe board developed for connecting nodes with PCIe external cable. The interface board for TCA is the PCI Express Adaptive Communication Hub version 2 (PEACH2) board, which uses an FPGA chip referred as the PEACH2 chip.

A. PCI Express Direct Communication Link (PEARL) and Its Hub Chip (PEACH)

We have proposed a direct communication link using PCIe as a concept, referred to as the PCI Express Adaptive and Reliable Link (PEARL) [5].

PCIe is the standard serial interface for attaching the device to a PC [6], and various I/O devices such as Ethernet, InfiniBand, and the GPU, are connected to a host via PCIe. In PCIe, multiple lanes can be used to expand the bandwidth. The expression "x n" denotes that n lanes are bundled in a link, for example, x4, x8, and x16 denote links with four, eight, and sixteen lanes, respectively. In the present version, PCIe Gen 3 has three lane-speed modes: 2.5 GHz, 5 GHz, and 8 GHz.

Originally, PCIe performs primarily memory read/write operations between the Root Complex (RC) on the host side and the Endpoint (EP) on the device side. However, the actual behavior is simply point-to-point bidirectional packet communication. Therefore, we used fast serial packet communication on the PCIe link for inter-node communication as the PEARL, and a PEACH (PCI Express Adaptive Communication Hub) chip containing four PCIe ports was developed as a kind of router chip for the purpose of realizing the PEARL[7], [8].

In PCIe, RC cannot communicate with another RC directly while the ordinary CPU serves only RC functions. Therefore, the PEACH chip mediates PCIe communication between nodes via PEARL by attaching to the node as a PCIe device.

On the other hand, in the conventional GPU cluster, GPU-to-GPU communication over nodes requires data copies to be performed in at least three steps. For example, when GPU-A on Node-A transfers the data to GPU-B on Node-B, the following data copies are required:

1) Copy from the memory in GPU-A to the memory in Node-A through PCIe.
2) Copy from the memory in Node-A to the memory in Node-B through the interconnect.
3) Copy from the memory in Node-B to the memory in GPU-B through PCIe.

In this case, if the interconnect is replaced by PEARL, and all of the memory copies are completed using the PCIe packet, which allows the PCIe packets to be relayed without overhead.

B. Overview of the PEACH2 chip

Realizing HA-PACS/TCA based on PEARL technology, we have been developing the PEACH2 chip. The PEACH2 chip provides fast operation for the relay function of the PCIe packets and sophisticated DMA functions, such as the chaining mechanism by hard-wired logic.

We implement the PEACH2 chip using a field programmable gate array (FPGA) for flexible control and prototyping. In recent years, a number of FPGA devices contain hard IPs for the PCIe interface, and we introduce Altera's Stratix IV GX FPGA, which includes four PCIe Gen2 x8 ports as the hard IP [9]. While the FPGA is restricted with respect to the operation frequency and the latency, the FPGA can fully support the performance with four PCIe Gen2 x8 ports and is suitable for experimental systems such as TCA because the logic can be flexibly changed, which allows the function to be improved and various new features to be added.

In the PEACH2 chip, four PCIe Gen2 ports are allocated, each of which has eight lanes, i.e., each port can transfer data at up to 4 Gbytes/sec. One of the four PCIe ports is dedicated to the interface between the CPU and the PEACH2 chip. Therefore, the other three ports can be used to connect the nodes.

C. Configuration of the TCA Node with Direct Communication between PEACH2 and GPUs

Figure 2 shows a block diagram of the communication in HA-PACS/TCA. Four GPUs are directly attached into the PCIe x16 slots in the conventional manner. Although the PEACH2 board is also inserted into another PCIe x8 slot, in this case, all of the devices, including GPUs and the PEACH2 board, also share a single PCIe address space, and the PEACH2 board can directly communicate with GPUs via the PCIe switch embedded in the CPU socket.

Accessing the GPU memory from the other devices is normally prohibited. However, a technology called GPUDirect Support for RDMA allows direct memory access through the PCIe address space under a CUDA 5.0 or above environment with a Kepler-class GPU [10]. Once this feature enables the GPU memory at page granularity to be mapped to the PCIe address space, the devices on the same PCIe bus can access the memory directly[1]. In practice, since the direct access over QPI between sockets degrades the performance significantly, we assume that PEACH2 only accesses GPU0 and GPU1 in Figure 2.

Since this configuration is similar to that of the HA-PACS base cluster, with the exception of the PEACH2 board, an accurate comparison can be performed in order to investigate the advantages of TCA. On the other hand, the I/O-rich platform, which provides 80 PCIe lanes by means of two Xeon E5 CPUs, is required for the compute node of the TCA environment.

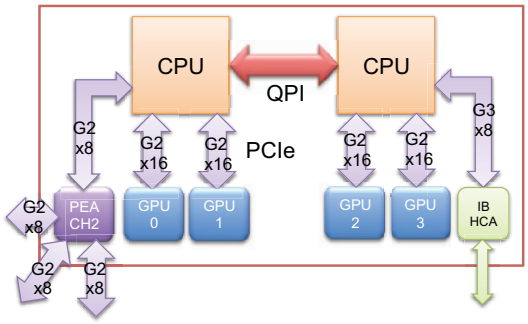

Figure 2. Block Diagram of the Computation Node in HA-PACS/TCA

D. Design of the PEACH2 chip

As mentioned above, four PCIe Gen2 x8 ports are equipped, and, for convenience, these ports are labeled North (N), East (E), West (W), and South (S). Port N is always

[1]We began developing such a function in cooperation with NVIDIA under a non-disclosure agreement and decided to introduce this technology to the PEACH2 chip after NVIDIA released the GPUDirect Support for RDMA technology.

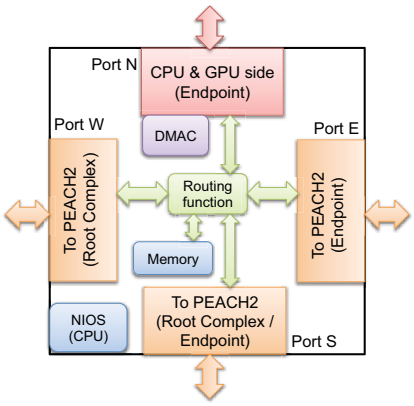

Figure 3. Overview of the PEACH2 chip

dedicated to the connection to the host CPU in order to be treated as the ordinary PCIe device. Ports E and W are expected to form the ring topology by connecting to each other. Since a PCIe link must be configured between RC and EP, the role of Ports E and W are fixed to EP and RC, respectively, in our design. Port S is selectable as RC or EP and is used to combine two rings by connecting to Port S on the peer node. Currently, different configuration images for the FPGA are prepared for switching the role of Port S. However, dynamic switching for the role of the port will be implemented because the partial reconfiguration for PCIe IP is available in this FPGA.

The DMA controller has a chaining mechanism that continuously operates multiple DMA requests with a single issue. This helps to improve the stride access caused by multidimensional array data or discontinuous access. As the packet buffer, the embedded memory in the FPGA and DDR3 SDRAM are available. DDR3 SDRAM is also used for the main memory of the controller, which is described in the following.

The PEACH2 chip also includes Altera's NIOS processor as a micro controller. The controller works only to monitor and manage PEARL, except for the packet transfer. Thus, a small, low-power controller is sufficient. In addition to the components shown in this figure, Gigabit Ethernet and RS-232C are equipped for communication with the NIOS processor.

E. PCIe Address Mapping and Routing Method

In order to possibly reduce the routing overhead, the routing function in the PEACH2 chip is designed so as to only check the destination address without table search or address conversion in order to decide the destination port.

Figure 4 shows an example of the address mapping technique in the TCA sub-cluster. Although the PCIe address has a 64-bit address space, the host and the devices attached to the host use only a small, limited area of the address space, and most of the address space is unused. Thus, PEACH2 reserves a relatively large address region (current implementation is 512 Gbytes)[2]. The address region is split equally as the aligned address to every node contained in the TCA sub-cluster. Furthermore, each split region is again divided into the aligned address block among two GPUs, the host, and the internal region of PEACH2, which can be accessed under the TCA environment in a node. The address offset information for each node can be commonly shared by every node within the TCA sub-cluster, and the unified address space in the entire TCA sub-cluster can be emulated by setting each PEACH2 chip appropriately. At the beginning of communication, the packet including the destination memory address of the target device in the packet header is sent to the PCIe bus, and the PEACH2 chip at the peer of the PCIe link receives the packet. Since these addresses are aligned, the receiving PEACH2 can rapidly decide the destination only by comparing the upper bits of the destination address. For the routing, it is necessary to assign an address range to the destination port. Figure 5 shows an example of the case of ring topology consisting of four nodes. In the routing mechanism, the control registers for the address mask, the lower bound, and the upper bound are prepared, and the destination port is statically decided by checking the result from the AND operation with the address mask.

Only if the PEACH2 receives the packet for the host to which the PEACH2 board is attached from the adjacent node, the address conversion from the global address space within the TCA to the internal address in the host is necessary at Port N. Actually, the base address of the PEACH2 chip and the address offset for the specified device are added to or subtracted from the destination memory address in the packet header.

F. Communication by PEACH2

PEACH2 provides two types of communication: PIO and DMA. On the PEACH2 chip, memory access to a remote node is restricted to Memory Write Request only on PCIe, i.e., PEACH2 supports only RDMA put protocol, because implementation for receiving the read reply packet (Completion with Data on PCIe protocol) is difficult and good performance cannot be expected for the read or get protocol. For Port N, the memory read access is allowed to read data from the host memory and the GPU memory in the local node.

1) PIO communication: PIO communication is useful for the short message transfer, and, as mentioned above, the CPU can only perform the store operation to remote nodes. The PCIe region assigned to the PEACH2 chip can be mapped through the device driver for the user space,

[2]The address region is set to the base address register (BAR) at boot time. In fact, the BIOS must be able to assign such large address regions. Currently, only a few motherboards can support the PEACH2 board.

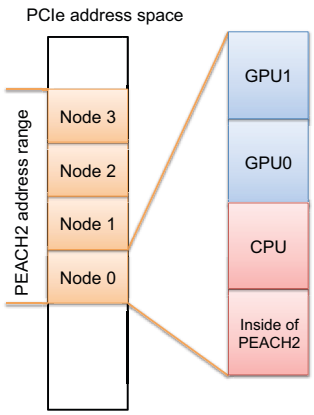

Figure 4. Address Mapping in the TCA Sub-cluster

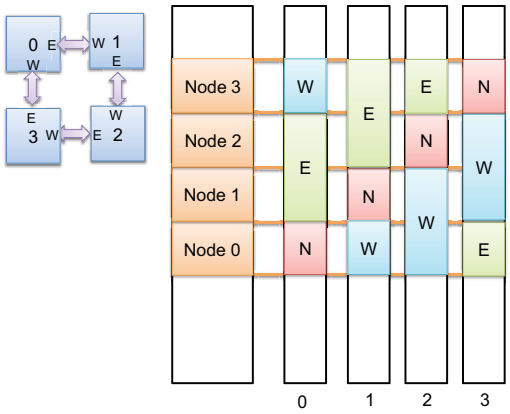

Figure 5. PEACH2 Routing Setting for the Output Port in Each Node (Ring with Four Nodes)

i.e., mmaped, and a user program can seamlessly perform RDMA write access according to an ordinary store instruction to the mmaped area.

2) DMA communication: A DMA controller is embedded in the PEACH2 chip. The DMA controller has the chaining DMA function in order to enable packet transfer with maximum performance. Implementation of the chaining DMA function is partially used on the IP contained in the PCIe reference design by Altera [11]. In this function, multiple DMA requests as the DMA descriptors are registered in the descriptor table in advance, and DMA transactions are then operated automatically according to the DMA descriptors by hardwired logic once the DMA descriptor table is activated.

G. PEACH2 Prototype Board

Figure 6 shows the PEACH2 prototype board. The board size is the full size of the PCIe board specification [12] (106.7 cm $\times$ 312 mm), and the board includes an edge connector with Gen2 x8 (for Port N) and three PCIe external cable ports (for Ports E, W, and S) at the side. In our design,

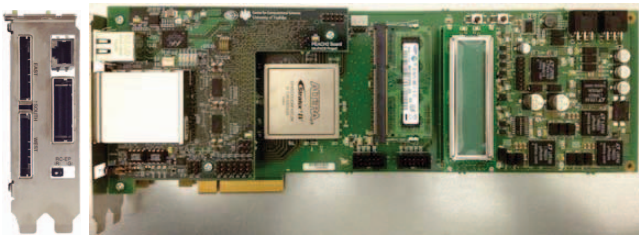

Figure 6. Photograph of PEACH2 Prototype Board (Side View(Left), Top View(Right))

Ports E and W use a x8 connector, and Port S uses a x16 connector with only eight lane data signals in order to relay the port configuration between RC and EP by sideband signals. A Stratix IV GX FPGA by Altera is mounted at the center of the board, and a DDR3 SDRAM SODIMM is also attached to the board. The greater part of right half of the board is occupied by the regulators generating the multiple voltage power for the FPGA. Port S is implemented on the sub-board, and the repeater chips for the PCIe signal is equipped with the sub-board in order to compensate the signal degradation caused by the connector between the main board and the sub-board.

The greater part of the PEACH2 chip operates at a clock frequency of 250 MHz, which is the operating clock frequency of the PCIe Gen2 x8 logic block.

H. Programming Interface for TCA

The Programming Interface for TCA is based on the CUDA parallel programming environment. CUDA 4.0 and later versions offers the Unified Virtual Address (UVA), which enables communication among the host and GPUs that share the same PCIe switch or the same CPU socket. This feature is referred to as GPUDirect Peer-To-Peer Transfers and Memory Access. We expand this feature to the direct communication among GPUs over the node on TCA. In the TCA sub-cluster, the node ID for each node is assigned beforehand, and then the user can specify the target node ID as well as the GPU ID.

For example, in the CUDA environment, the *cudaMemcpyPeer()* function offers GPU-to-GPU memory copy within a node. In the TCA sub-cluster, a function similar to cudaMemcpyPeer should be available for the target node ID in addition to the GPU IDs in cudaMemcpyPeer.

CUDA-like APIs are very useful for expanding existing CUDA applications to the TCA sub-cluster. Moreover, a series of bulk transfers, such as block transfer and block-stride transfer, are effective by using the chaining DMA mechanism.

IV. Preliminary Performance Evaluation

In this section, we evaluate the basic performance of the PEACH2 chip using the PEACH2 prototype board and current implementation for FPGA logic design.

Table II
TEST ENVIRONMENT FOR PRELIMINARY PERFORMANCE EVALUATION

Hardware	
CPU	Xeon-E5 2670 2.6 GHz × 2
Memory	DDR3 1600 MHz × 4 ch, 128 Gbytes
Motherboard	(a) SuperMicro X9DRG-QF (b) Intel S2600IP
GPU	NVIDIA K20 2496 cores, 705 MHz
Memory	GDDR 5 2600 MHz, 5 Gbytes
PEACH2 prototype board	16 layers (main) + eight layers (sub)
FPGA	Altera Stratix IV GX 530, 290 1932 pin (EP4SGX{530,290}NF45C2N)
PEACH2 Logic	version 20121112
Software	
OS	Linux, CentOS 6.3 kernel-2.6.32-279.{9,14,19}.1.el6.x86_64
GPU Driver	NVIDIA-Linux-x86_64-304.{51,64}
Programming Environment	CUDA 5.0

Table II indicates the test environment for the evaluation. We develop two device drivers: the PEACH2 driver for controlling the PEACH2 board and the P2P driver for enabling GPUDirect Support for RDMA.

A. DMA Basic Performance

First, we evaluate the basic transfer performance within a node using the PEACH2 chip. In this measurement,

1) the PCIe performance on the FPGA and
2) the performance of the DMA controller (DMAC)

can be evaluated.

The clock counter embedded in the CPU (TSC) is used for this measurement. Once the clock counter is checked just before DMA start, the clock counter is checked again in the interrupt handler generated by the completion from the DMAC in the PEACH2 driver.

Here, we assume that the direction of the DMA is defined from viewpoint of the PEACH2 chip, for example, a DMA write indicates a transfer from PEACH2 to CPU/GPU.

1) Transfer between PEACH2 and CPU within a Node: Here, we evaluate the transfer performance between PEACH2 and the CPU within a node. A DMA buffer is prepared in the PEACH2 driver beforehand and is available for the source or destination of DMA transfer.

Figure 7 shows the results for 255 DMA writes and DMA reads to the local CPU as the target using the DMA buffer in the device driver by chaining. In order to aggregate the number of DMA requests, the impact of the overhead for retrieving the DMA descriptor table can be reduced.

As a result, the maximum performance of the DMA transfer through PCIe reaches 3.3 Gbytes/sec in the case of a data size of 4 Kbytes.

The theoretical peak performance considering the PCIe packet header can be calculated as follows. PCIe Gen2 uses a 5-GHz signal and provides 4 Gbytes/sec with eight lanes

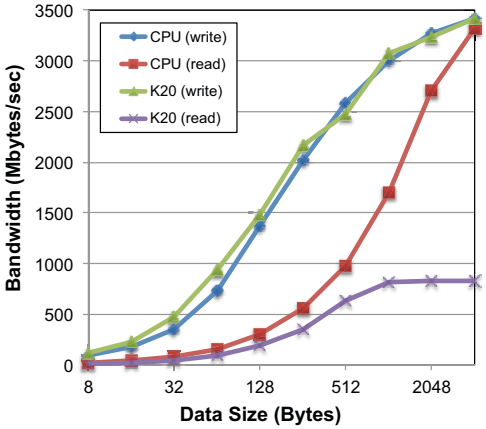

Figure 7. Data Size vs. Bandwidth between PEACH2 and the CPU/GPU (DMA 255 times)

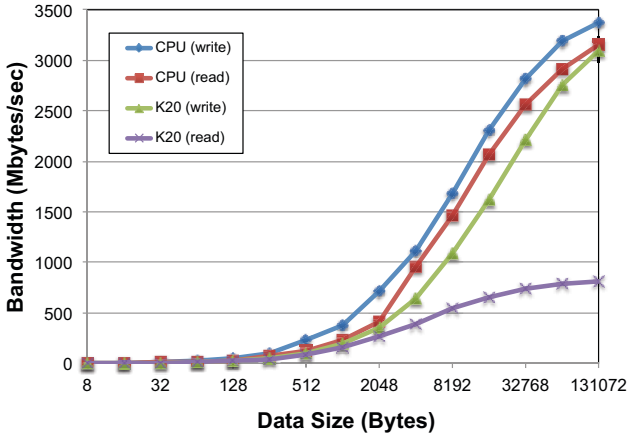

Figure 8. Data Size vs. Bandwidth between PEACH2 and the CPU/GPU (single DMA)

due to 8b/10b encoding for the embedded clock and error detection. In the experimental environment, the maximum payload size is 256 bytes. In other words, for every 256 byte payload, a Transaction Layer header of 16 bytes, a sequence number of 2 bytes on the Data Link Layer, an LCRC for error check of 4 bytes, and the start and stop frames of 1 byte each on the Physical Layer, are located in a packet. Thus, the theoretical peak performance is given by:

$$4 \text{ Gbytes/sec} \times \frac{256}{256+16+2+4+1+1} = 3.66 \text{ Gbytes/sec}.$$

The experimental results achieve a performance of 93% of the theoretical peak performance by DMA write for a message size of 4 Kbytes and indicate that PCIe IP and the chaining DMA controller on the PEACH2 chip provides sufficient performance. The performance of DMA write is better than that of DMA read. This is because that DMA

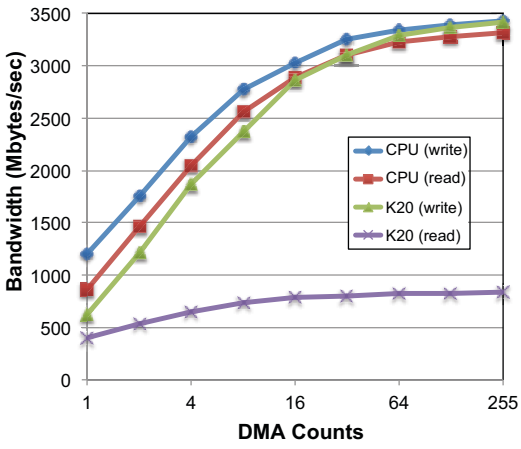

Figure 9. Number of DMA Requests vs. Bandwidth between PEACH2 and the CPU/GPU (fixed data size at 4 Kbytes)

read request requires a reply containing read data, whereas the DMA write request can issue the next message without waiting for the completion of this request. However, the performance of DMA read for a message size of 4 Kbyte is approximately the same as that of DMA write.

Next, in order to measure the overhead for transferring the descriptor table, the DMA performance for a single request is examined. The results are shown in Figure 8. In this case, the performance is severely degraded compared to the case of 255 burst transfers. Since retrieving the descriptor table is the dominant factor in performance degradation, the DMA function without a descriptor is also desired for relatively small amounts of data, i.e., several hundreds or thousands of bytes. Figure 9 shows the results obtained by varying the number of DMA requests from 1 to 255 with the data size fixed at 4 Kbytes. DMA transfer including four requests achieves approximately 70% of the maximum performance. In addition, the results for two or more requests are approximately the same as that for 8 Kbytes or more in Figure 8. These results indicate that, if the actual transfer amount is the same, the DMAC has approximately the same performance despite the number of descriptors.

2) Transfer between PEACH2 and GPU within a Node: In the following experiment, we measure the DMA read/write performance to the GPU memory, which is allocated by CUDA API using PEACH2. The CUDA driver API is used throughout this experiment.

In order to access the GPU memory, according to GPUDirect Support for RDMA [10], the following steps are required:

1) GPU memory is allocated by a *cuMemAlloc()* function, such as `cuMemAlloc(&addr, size)`.
2) The token to obtain access permission for the specified GPU memory is obtained by a *cuPointerGetAttribute()* function, such as `cuPointerGetAttribute(&token, CU_POINTER_ATTRIBUTE_P2P_TOKENS, addr)`
3) A part of the GPU memory is pinned to the PCIe address space with the token by the P2P device driver.
4) The other devices can access the GPU memory using the assigned PCIe address.

As mentioned above, we have developed a P2P driver for Step 3, and PEACH2 can access the GPU memory correctly. The difference from the previous experiment is that the target address for DMA request is only changed to the PCIe address mapped in the Base Address Register (BAR) for GPU memory.

The results are shown in Figures 7 through 9 along with the results from the CPU, for the case of 255 burst transfers for various data sizes, a single DMA transfer, and a fixed data size of 4 Kbytes for various burst counts. As a result, the performance of the DMA write to the GPU memory is approximately the same as that of the CPU memory, and these results indicate that direct communication between PEACH2 and the GPU memory bypassing the CPU can achieve the maximum bandwidth for PCIe.

In contrast, the maximum DMA read performance is only 830 Mbytes/sec. The reason for this is considered to be related to the address conversion mechanism in order to map the PCIe address space within the GPU. However, the details are unclear. Moreover, the performance of DMA write access to the GPU on another socket over QPI is severely degraded by up to several hundred Mbytes/sec, although we omit the details from the figures and P2P access through PCIe over QPI should be still prohibited.

B. Latency and Bandwidth among Different Nodes

Next, we investigate the latency for the routing in the PEACH2 chip and transfer through the PCIe link.

1) Latency Measurement: In order to strictly measure the latency among the PEACH2 chip, two PEACH2 boards are attached to a single node, as shown in Figure 10. In this experiment, several store operations from the CPU by PIO transfer are performed, and the round-trip time is calculated based on the results.

The packet transfer in this configuration proceeds, and the latency is measured as follows:

1) Routing information is appropriately set to the control register in PEACH2.
2) Read the clock counter in the PEACH2-A driver.
3) Store 4-byte data into the region assigned to PEACH2-B within the PCIe address space of PEACH2-A.
4) A PCIe packet is transferred automatically from PEACH2-A to PEACH2-B via a PCIe external cable.
5) PEACH2-B writes the data from the packet to the specified address.
6) The PEACH2-A driver polls the address written by PEACH2-B. If the data in the address is changed,

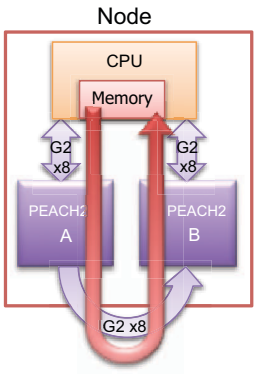

Figure 10. Configuration of the Loopback Test Using PIO

the driver reads the clock counter and calculates the difference from the counter value of Step 2.

As a result, the transfer latency is 782 nsec using the current FPGA logic implementation. The latency of InfiniBand FDR with PCIe Gen3 x8 is announced as less than 1 μsec [13]. Therefore, the latency of PEACH2 is approximately the same or slightly less than that of InfiniBand. Moreover, there is room to improve the FPGA logic for PEACH2.

2) Bandwidth among Different Nodes: Finally, we investigate the bandwidth from PEACH2 to the CPU memory on an adjacent node using DMA write. Figure 11 indicates the configuration of this experiment.

Figure 12 shows the results for the bandwidth obtained for 255 DMA transfers to the CPU and GPU on an adjacent node. In this figure, the CPU (write) and CPU (read) lines are the same results shown in Figure 7 obtained from the experiment in Section IV-A1, and are shown for comparison. The bandwidth to the CPU memory decreases for the small data size due to the latency for transfer between PEACH2. However, the bandwidth at 4 Kbytes is approximately the same as the bandwidth within a node. On the other hand, the bandwidth to the GPU memory is approximately the same as the bandwidth within a node. The reason for this is unclear, but the GPU is assumed to be of sufficient size for the request queue from PCIe and it is assumed that PEACH2 can send the PCIe packet continuously, which is similar to the case of the local node.

In the current DMAC in the PEACH2 chip, the internal memory of PEACH2 must be specified as the source address on DMA write and as the destination address on DMA read. For this reason, in order to send the data in a local node to a remote node, two phase operations are required. As the first phase, data must be stored in the internal memory by DMA read, and in the second phase, data in the internal memory is written to the CPU or GPU memory on the other node. However, since this procedure seriously impacts the performance, we are developing a new DMAC, which operates both the read request from the memory on the local

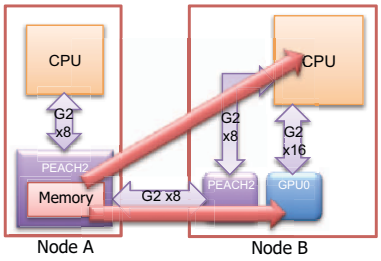

Figure 11. Configuration of the Remote DMA Test from PEACH2 on Node A to the CPU/GPU on Node B

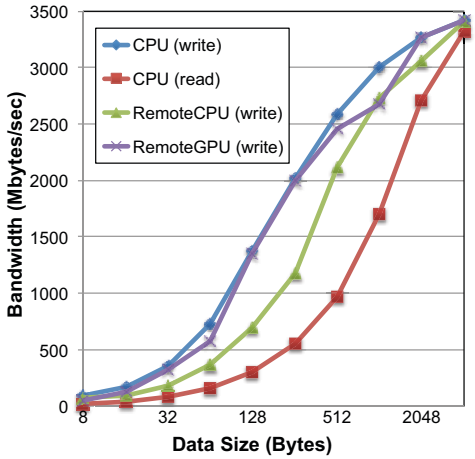

Figure 12. Data Size vs. Bandwidth between PEACH2 and CPU/GPU on an Adjacent Node via PEACH2 (DMA 255 times)

node and the write request to the memory on the remote node simultaneously in a pipeline manner. This enables the data transfer from a certain node to be specified by only a DMA descriptor within the TCA sub-cluster.

V. Related Research

The non-transparent bridge (NTB), which is embedded in the PCI-E switch, allows inter-node communication by means of a special function[14]. Several products based on NTB capability have been released. The NTB technology appends the bridge function to a downstream port of the PCI-E switch. The bridge behaves as two different EPs by accessing either the specified downstream port or the other ports, and address translation is performed between the upstream port and the downstream port within the NTB. Thus, RC, which is connected to the downstream port with the NTB, can communicate with the other RCs via the NTB, and all of the EPs can communicate with each other. However, the NTB is not defined in the standard of PCI-E and is incompatible with the chip vendors of PCI-E switches. Furthermore, during the BIOS scan at boot time, the host must recognize the EPs in the NTB and disconnection of

the node causes a system reboot. On the other hand, the PEACH2 chip has independent PCIe ports, and the link state with the other node has no impact on the connection between the host and the PEACH2 chip.

APEnet+ is the original 3-D Torus network using the FPGA [15], [16]. APEnet+ enables GPU direct copy over the nodes using Fermi-architecture GPUs. However, APEnet+ uses QSFP+ as a cable, and PCIe is used only for the host connection.

As mentioned in Section III-C, the CUDA 5 programming environment for NVIDIA GPU provides the RDMA mechanism to the GPU memory, called GPUDirect Support for RDMA, with a Kepler-class GPU [3]. This mechanism enables zero-copy communication between the InfiniBand Host Adaptor and GPUs[17]. Although PEACH2 also uses the RDMA mechanism, PEACH2 can eliminate the overhead for protocol conversion from PCIe to the InfiniBand packet. Moreover, as mentioned in Section III-H, applications on the TCA sub-cluster do not rely on the MPI software stack. As a result, the overhead of MPI protocol stack can be eliminated.

VI. Conclusion

In the present paper, we introduced a novel interconnect for direct communication among accelerators over nodes, referred to as the TCA architecture. A newly developed PEACH2 chip using the FPGA, which includes four PCIe Gen 2 x8 ports, a DMA controller, and an embedded micro controller, was presented. We also developed a prototype board equipped with the PEACH2 chip. The results of a preliminary evaluation demonstrate that the PEACH2 chip has sufficient maximum performance with 93% of the theoretical peak performance and a latency between adjacent nodes of 782 nsec, and GPU-to-GPU direct communication based on TCA is feasible.

We have designed a production version of the PEACH2 board. Furthermore, we prepare an eight-node cluster as a TCA sub-cluster for the experiment and continue to implement the TCA function and to improve the performance. At the same time, coding, performance evaluation, and product run for large-scale GPU applications have already been performed using the HA-PACS base cluster. Based on these results, we will prepare an API for using TCA and implement full-scale scientific applications using TCA. In the fall of 2013, the HA-PACS/TCA cluster, which will include several dozen compute nodes (each of which has four GPUs, an InfiniBand host adaptor, and a PEACH2 board) will be introduced. This system will be able to demonstrate the effectiveness of the TCA architecture.

Acknowledgment

The authors would like to thank Dr. Dale Southard, Dr. Joel Scherpelz, and the other members of NVIDIA and NVIDIA Japan for their helpful discussion and assistance. The authors would also like to thank the members of the Project Office for Exascale Computational Sciences and the Project Office for Exascale Computing System Development, Center for Computational Sciences, University of Tsukuba. The present study was supported in part by the MEXT special fund program entitled "Research and Education on Interdisciplinary Computational Science Based on Exascale Computing Technology Development" and by the JST/CREST program entitled "Research and Development on Unified Environment of Accelerated Computing and Interconnection for Post-Petascale Era" in the research area of "Development of System Software Technologies for post-Peta Scale High Performance Computing."

References

[1] J. Dongarra, H. Meuer, E. Stromaier, and H. Simon. Top500 list. [Online]. Available: http://www.top500.org

[2] *NVIDIA Tesla Kepler GPU Computing Accelerators*, NVIDIA Corp. [Online]. Available: http://www.nvidia.com/content/tesla/pdf/Tesla-KSeries-Overview-LR.pdf

[3] *NVIDIA GPUDirect*, NVIDIA Corp. [Online]. Available: http://developer.nvidia.com/gpudirect

[4] *NVIDIA CUDA: Compute Unified Device Architecture*, NVIDIA Corp. [Online]. Available: http://developer.nvidia.com/category/zone/cuda-zone

[5] T. Hanawa, T. Boku, S. Miura, T. Okamoto, M. Sato, and K. Arimoto, "Low-power and high-performance communication mechanism for dependable embedded systems," in *Proceedings of 2008 International Workshop on Innovative Architecture for Future Generation Processors and Systems*, Jan 2008, pp. 67–73.

[6] *PCI Express Base Specification, Rev. 3.0*, PCI-SIG, Nov. 2010.

[7] S. Otani, H. Kondo, I. Nonomura, M. Uemura, Y. Hayakawa, T. Oshita, S. Kaneko, K. Asahina, K. Arimoto, S. Miura, T. Hanawa, T. Boku, and M. Sato, "An 80Gbps dependable communication SoC with PCI Express I/F and 8 CPUs," in *2011 IEEE International Solid-State Circuits Conference*, Feb 2011, pp. 266–267.

[8] S. Otani, H. Kondo, I. Nonomura, T. Hanawa, S. Miura, and T. Boku, "Peach: A multicore communication system on chip with PCI Express," in *IEEE Micro*, vol. 31, no. 6, 2011, pp. 39–51.

[9] *Stratix IV Device Handbook*, Altera Corp. [Online]. Available: http://www.altera.co.jp/literature/lit-stratix-iv.jsp

[10] *Developing A Linux Kernel Module Using RDMA For GPUDirect*, NVIDIA Corp. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/5_0/rc/docs/GPUDirect_RDMA.pdf

[11] *IP Compiler for PCI Express user guide*, Altera Corp., http://www.altera.com/literature/ug/ug_pci_express.pdf.

[12] *PCI Express Card Electromechanical (CEM) Specification, Rev. 2.0*, PCI-SIG, Apr. 2007.

[13] *ConnectX-3 VPI Product Brief*, Mellanox Technologies, Ltd. [Online]. Available: http://www.mellanox.com/related-docs/prod_adapter_cards/ConnectX3_VPI_Card.pdf

[14] J. Gudmundson, "Enabling multi-host system designs with PCI Express technology," PLX Technology Inc., May 2004. [Online]. Available: http://www.plxtech.com/products/expresslane/techinfo

[15] R. Ammendola *et al.*, "APEnet+: high bandwidth 3D torus direct network for petaflops scale commodity clusters," in *Journal of Physics*, ser. Conference Series, vol. 331, Part 5, no. 5, 2011.

[16] D. Rosetti *et al.*, "Leveraging NVIDIA GPUDirect on APEnet+ 3D torus cluster interconnect," May 2012. [Online]. Available: http://developer.download.nvidia.com/GTC/PDF/GTC2012/PresentationPDF/S0282-GTC2012-GPU-Torus-Cluster.pdf

[17] *Mellanox OFED GPUDirect*, Mellanox Technologies, http://www.mellanox.com/content/pages.php?pg=products_dyn&product_family=116&menu_section=34.