Performance Evaluation of PEACH3: Field-Programmable Gate Array Switch for Tightly Coupled Accelerators

Takahiro Kaneda,
Ryotaro Sakai,
Naoki Nishikawa,
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
kaneda@am.ics.keio.ac.jp

Toshihiro Hanawa
The University of Tokyo
5-1-5 Kashiwanoha, Kashiwa,
277-8589, Japan
hanawa@cc.u-tokyo.ac.jp

Chiharu Tsuruta,
Hideharu Amano
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
asap@am.ics.keio.ac.jp

An FPGA switching hub for tightly coupled accelerators (TCA) architecture called PEACH3 (PCI-Express Adaptive Communication Hub ver. 3) is evaluated and its communication speed is analyzed. PEACH3 connects a number of GPUs directly through PCI express Gen3x8 ports. The latency of inter-node GPU-GPU communication of PEACH3 was about 2.8 μ sec which is one third of that of CUDA API with MPI/Infiniband. The bandwidth was about 1.21 times of that of the previous version PEACH2, and 1.54 times of that with MPI/Infiniband for 512KB data transfer. Two application programs: BFS (breadth first search) and CG (conjugate gradient) were implemented with TCA IP and CUDA IP with MPI/Infiniband. The performance of BFS with PEACH3 was 1.16 times better than that with PEACH2, and 1.3 times better than that with MPI/Infiniband for a graph with $scale = 15$. In CG, for the small matrix (CLASS=S), the PEACH3 achieved 12% better performance than that with PEACH2 and 25% with MPI/Infiniband. However, since the bandwidth of PEACH3 with PCI gen3x8 is smaller than Infiniband with PCI gen3x16, the performance benefit was disappeared for CLASS=A matrix. Through the evaluation, it appears that if the data size is small, using TCA API with PEACH3 is advantageous even for intra-node communication.

Keywords

GPU, Cluster, TCA, PEACH3

1. INTRODUCTION

Heterogeneous clusters with GPUs have been the mainstream of high performance computing. A large number of systems with GPUs are registered in Top500 supercomputer ranking including TITECH's TSUBAME2.5[1], and Titan [2]. Such high performance systems are also expected to be used for recent emerging big data processing as well as numerical computation. For such big data processing, a high latency between GPUs communicating across nodes by indirect communication via the memory of the host CPUs often forms the bottleneck, since it is hard to hide the communication time in the calculation time.

Tightly Coupled Accelerators (TCA) architecture[3][4] has been proposed for direct data communication between accelerators connected to different nodes. TCA commonly uses PCI Express (PCIe) to connect a host CPU and accelerators in a node as a network that connects multiple nodes. A key component of TCA architecture is a switching hub called PEACH (PCI Express Adaptive Communication Hub) implemented on an FPGA. The 2nd version of PEACH using PCIe Gen2 x8 for communication has been used in a supercomputer HA-PACS/TCA[5]. However, the bandwidth is often insufficient because of the bottleneck at its out-of-date PCIe Gen2 ports. PEACH3 with PCIe Gen3 x8 ports on Stratix V FPGA was thus developed as a successor to PEACH2, but the performance evaluation has not been presented except for a simple preliminarily evaluation of the hardware[6][7]. Recently, high performance I/O subsystems which can connect accelerators more tightly have appeared.[8][9][10]. In IBM CAPI, a power service layer (PSL) on an FPGA is connected to a coherent accelerator processor proxy (CAPP) provided on POWER8, and a virtual memory space is formed between the CPU and accelerators. NVLink provided in a new generation GPU Pascal P100 can connect CPU-GPU and GPU-GPU directly. The NVIDIA DGX-1 which provides eight GPUs connected with NVLink got the first place in Green500. However, NVLink can be used only within a single node, and PCIe is still needed for the control of NVLink. APEnet+ also uses original high speed links for connection between boards, while each board is connected to the host through PCIe[10]. On the contrary, PEACH3 uses PCIe Gen3 commonly used in most systems for all interconnection, and thus, logically tight connection is easy to be introduced.

Here, the application performance of a system with PEACH3 was evaluated and compared with that of CUDA API and PEACH2. The contribution of this paper is as follows.

- Unlike the preliminarily evaluation presented in[6][7] the paper includes the performance evaluation of enough optimized communication primitives of PEACH3.
- The performance evaluation of practical application programs is presented and influences by the latency and bandwidth of PEACH3 are analyzed.
- For inner-node and inter-node communication, guidelines for programmers how to use APIs are shown.

The rest of paper is organized as follows: Section 2 introduces TCA and its switching hubs PEACH2/3. Section

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2017) Bochum, DE, June 7-9, 2017.

3 shows application programs implemented for evaluation with their communication pattern. The evaluation results are presented and analyzed in Section 4 with guidelines for using TCA API. Section 5 concludes this paper.

2. PEACH3

PEACH3 is a PCIe Gen3 switch for a TCA implemented on a common FPGA, Altera's Stratix V. It is a successor of the previous version PEACH2 using PCIe Gen2.

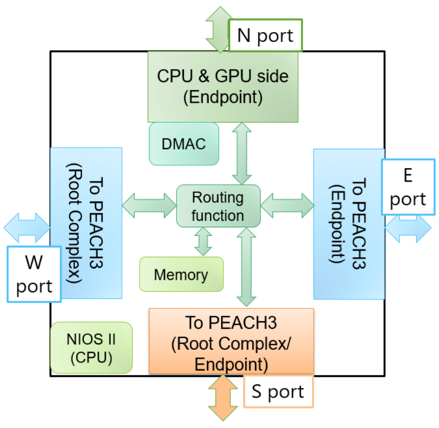

Figure 1: Block diagram of PEACH3 FPGA

As shown in Figure 1, the PEACH3 FPGA provides four PCIe Gen3 x8 ports: N, W, E, and S. Each ports is assigned to the endpoint (EP) or the root complex (RC). In the PCIe, the EP port must be connected to the RC port. Port N is for the card edge connector to the host CPU; therefore, it is the EP port. The port W as RC and the port E as EP are used to form a ring network by connecting corresponding ports of neighbors, and the the port S is RC/EP selectable and is used to form a double ring network of S/S' links.

The number of nodes in a TCA sub-cluster is limited to 16, and the double ring network is formed to connect them. The memories of all nodes connected with PEACH3 are mapped onto a single 512 Gbytes address region for packet routing. The address region is split equally among every node in the TCA sub-cluster; thus, 32 Gbytes are assigned to each node. The address of each node is divided for the CPU, GPU, and PEACH3. The PCIe packets are routed based on the 64-bit address according to the memory map. The PIO data transfer and several block data transfers are supported.

Currently, only the Stratix V GXA7 sub-family with an advanced 28-nm process supports four PCIe Gen3 x8 IPs. Therefore, we used EP5SGXA7N3F45C2 with more logic gates than EP4SGX530NF45C2 used in PEACH2. Similar to the situation in which PEACH2 was designed, there are almost no other choices for the FPGA in PEACH3. There is another benefit to using Stratix V. This family supports partial reconfiguration, which is not supported in Stratix IV. By using this function, the unused logic can be used for accelerating applications easily [11][12] .

The functions of PEACH3 is fully compatible with PEACH2. The only additional function is initiating the DMA from NIOS II or user-defined logic. In PEACH2, the DMA is only initiated by the host CPU, and this restriction sometimes adds extra overheads of tasks with NIOS II or user-defined logic. As a result, the software for PEACH2 runs on PEACH3 without any modification. The internal network of PEACH3 using Avalon ST was extended to 256 bits from 128 bits in PEACH2. This allows use of the same 250 MHz clock used in the PCIe Gen3 hard IP in all internal logic in PEACH3. The DMA logic was modified so that it is initiated from NIOS II and user-defined logic.

Table 1 summarizes the specifications of PEACH2 and PEACH3.

Table 1: Specification of PEACH2/PEACH3

	PEACH2	PEACH3
FPGA	Stratix IV GX EP4SGX53 -0NF45C2	Stratix V GX EP5SGXA7 -3F45C2
Process	40nm	28nm
LEs	531K	622K
PCIe Port	PCIe Gen2 x8	PCIe Gen3 x8
PCIe bandwidth	4Gbyte/sec	7.9Gbyte/sec
FPGA frequency	250MHz	250MHz
Internal bus	128bit	256bit
Utilization ratio	22%	38%
DRAM	DDR3 512Mbyte	DDR3 512Mbyte

The utilization ratio of PEACH3 was increased since the internal bus was extended to 256bits. Although there still remains room to implement user-defined logic, the 250MHz clock frequency is almost the upper limit of the internal network of PEACH3. This difficulty in implementation is the major reason why the upgrade from PEACH2 to PEACH3 is rather conservative.

3. IMPLEMENTATION

We implemented two applications, breadth-first search (BFS) and conjugate gradient method (CG). The former is non-numerical application and the latter is a typical numerical application. In order to analyze the impact of communication to application performance, the size and direction of communication required in each application are summarized.

3.1 TCA API

For application programmers, the following TCA APIs are used. First, memory is allocated by using *tcaMalloc*. Then, with *tcaCreateHandle*, *tcaHandle* is generated. For communication, it is exchanged and shared in all nodes. For DMA transfer, a DMA descriptor is created by using *tcaDescNew*. The sender node registers the DMA request by using *tcaDescSetMemcpy* to the descriptor, and the descriptor is set to a DMA with *tcaDescSet*. The sending nodes then starts the communication by the *tcaStartDMADesc*, and *tcaWaitDMARecvDesc* synchronizes the sender and the receiver. Although the initialize of the DMA requires several steps, the communication can be done only with *tcaStartDMADesc*.

3.2 Breadth-first search (BFS)

The BFS is a graph search algorithm frequently used in big data processing, and we adopted Graph500, a well known benchmark[13]. In Graph500, the BFS is executed 64 times and its harmonic mean of the number of searched edges in a second TEPS (traversed edges per second) is evaluated as a performance measure. A graph has 2^{scale} vertexes and $2^{scale} \times edgefactor$ edges.

For parallel execution, a level-synchronized BFS is used. The level means the depth from the root vertex. Threads start searching from the Level 1 (the distance from root vertex is 1). When all the threads finish searching for the vertexes with Level 1, the search for the next level, Level 2 starts. That is, synchronization is done at each level. There are two search methods: top-down which searches un-visited vertexes from the target vertex and bottom-up which searches the target vertex from un-visited vertexes. Here, we adopted bottom-up BSF which can be simply implemented. Vertices are equally divided by the number of GPUs and allocated to each GPU.

Three types of communication are needed in the level-synchronized BFS. First, each GPU has a scalar value for a flag indicating whether all adjacent vertexes have been searched. It is transferred from each GPU to the host CPU, when the search of each level is completed. Then, the data are concentrated to a node with node-to-node communication for judgment of processing termination. Here, the communication for the flag is called COMM_FLAG. It includes GPU-CPU communication and inter-node communication.

Second, each GPU has a vector for storing visited neighboring vertexes. Since the vector is used for searching the next level, it must be exchanged between each GPU also when the search of each level is completed. The communication is called COMM_OUT. Since it must be communicated between GPUs, TCA API is used for inter-node communication and CUDA API is used for inner-node communication. A large size of data related to *scale* of the graph are exchanged between GPUs.

When all vertexes are visited, each GPU has a part of final results to be merged. Here, the third communication for merging them is called COMM_TREE. Like COMM_FLAG, it requires GPU to CPU, and inter-node communication.

The detail of communication when $|V|$ vertexes are searched with $|P|$ GPUs is shown in Table2.

Table 2: The detail of communication (BFS)

	direction	data size	times/BFS
COMM_FLAG	GPU $\Rightarrow$ CPU	4 Bytes	$4 \sim 7$
COMM_OUT	GPU $\Rightarrow$ GPU	$\frac{4*\lvert V\rvert}{\lvert P\rvert}$ Bytes	$4 \sim 7$
COMM_TREE	GPU $\Rightarrow$ CPU	$\frac{8*\lvert V\rvert}{\lvert P\rvert}$ Bytes	1

3.3 Conjugate gradient method (CG)

The CG method is an iterative method for solving large scale linear equations. Here, CG in NAS Parallel Benchmarks (NPB) [14], a small set of programs to evaluate the performance of parallel supercomputers is used. In CG, a sparse-matrix vector multiplication (SpMV), a vector inner product (DOT), and a vector addition (AXPY) are operated repeatedly. When each GPU computes a part of the matrix in parallel, three types of communication are needed.

First, after local SpMV is done on each GPU, it exchanges the vector data with each other for getting the results of SpMV. Here,this communication is called COMM_SPMV.

Second, COMM_DOT is the communication for reduction computation the for results of the local inner products. Each CPU sends the scalar value of the local reduction result. Since this communication is done between CPUs, MPI/Infiniband is used.

Finally, the vector data each GPU has are exchanged and this communication is called as COMM_EXCH.

The table3 shows details of communication when using $|P|$ GPUs and $N \times N$ matrix.

Table 3: The detail of communication (CG)

	direction	data size	times
COMM_SPMV	GPU $\Rightarrow$ GPU	$\frac{8N}{\lvert P\rvert}$ Bytes	$\sqrt{\lvert P\rvert}$
COMM_DOT	CPU $\Rightarrow$ CPU	8 Bytes	$\sqrt{\lvert P\rvert}$
COMM_EXCH	GPU $\Rightarrow$ GPU	$\frac{8N}{\lvert P\rvert}$ Bytes	$\sqrt{\lvert P\rvert}$

4. EVALUATION

4.1 Evaluation Setup

The evaluation setup is shown in Table 4. For evaluation of PEACH3, two nodes, each of which provides CPU, GPU and PEACH3, are connected with a 50-cm PCIe Gen3 cable as shown in Fig. 2. W port and E port of the PEACH3 boards are connected to each other. For comparison, HA-PACS/TCA owned by the University of Tsukuba is used for measurement performance of PEACH2 and Infiniband.

Table 4: Evaluation Setup

	Machine with PEACH3	HA-PACS/TCA
CPU	Intel Xeon E5-2680 v2(Ivy Bridge-EP)	
Clock	2.8 GHz	
CPU Memory	32 GB, DDR3	128GB, DDR3
Memory Clock	1600MHz	1866MHz
GPU	Tesla K40m	Tesla K20x
TCA board	PEACH3	PEACH2
PCIe(TCA)	Gen3 x8	Gen2 x8
Infiniband	FDR 1rail	QDR 2rail
gcc	4.4.7	
CUDA	7.0	
NVIDIA Driver	352.39	367.55
icc	17.0.1	15.0.3
MVAPICH2	GDR-2.1	GDR-2.1rc

4.2 Intra-node communication with APIs

In TCA architectures, application programs can use both CUDA API and TCA API for intra-node communication. First, we evaluated the latency and bandwidth of each API.

Fig. 3 shows the latency versus data size of an intra-node data transfer. For data size smaller than 2KB, the latency using PEACH3 is about a third of the case with CUDA API in CPU-GPU data transfer. For GPU-GPU transfer, the latency is about 66% of the case with CUDA API. It comes from the large start up latency in the CUDA API can be omitted in TCA API. However, when the data size is more than 8KB for GPU-GPU, 32KB for GPU-CPU and 64KB for CPU-GPU, CUDA API which can use PCIe Gen3x16 unlike PEACH3 using PCIe Gen3x8 becomes advantageous.

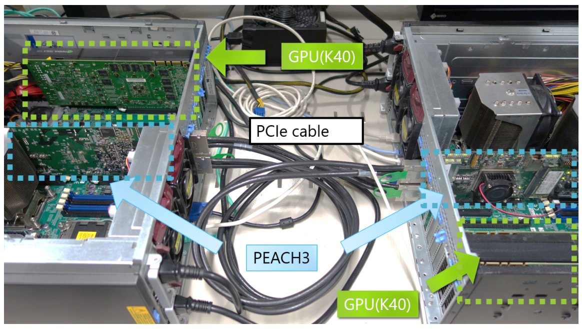

Figure 2: A system used in the evaluation

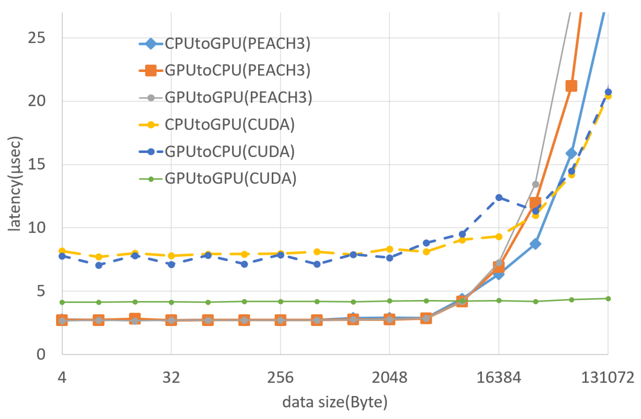

Figure 3: Latency vs. Data size of intra-node communication

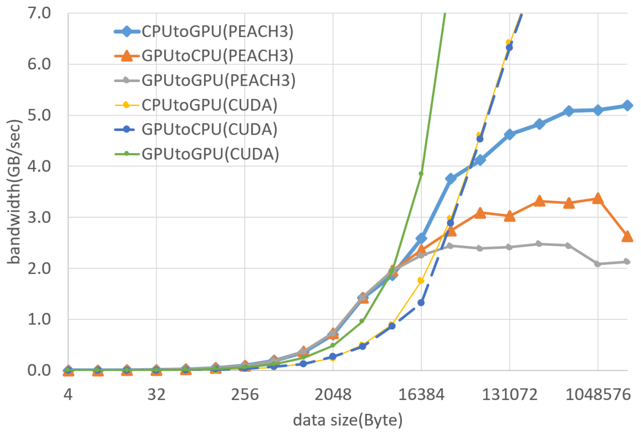

Figure 4: Bandwidth vs. Data size of inner-node communication

Fig. 4 shows the bandwidth versus data size of an inner-node data transfer. The bandwidth of TCA API with PEACH3 is advantageous when the data size is small. However, for large data size the bandwidth of CUDA API is rapidly increased, while TCA API with PEACH3 is saturated. This results show that even for inner-node communication, if the transferring data size is less than 8KB, using TCA API is advantageous. However, for a large data size communication, CUDA API should be used.

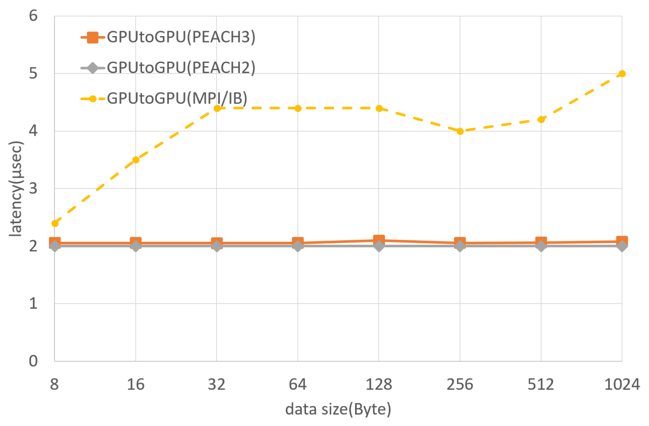

Figure 5: Latency vs. Data size of inter-node communication

4.3 Inter-node communication with APIs

In TCA architectures, application programs can use both infiniband (IB) based MPI and TCA API for inter-node communication. Fig. 5 shows the latency versus data size of GPU-GPU communication attached to the different nodes. TCA API achieves 2μsec latency both the case of PEACH2 and PEACH3. We had been afraid that the latency of PEACH3 might be stretched when the serial data is expanded to 256bit in the FPGA, but the latency is almost the same as PEACH2 which uses 128bit inner bus. Since the latency is less than a half of the case with MPI/IB, TCA APIs are advantageous for latency sensitive inter-node communication.

The bandwidth versus data size of GPU to GPU communication connected to the different node is shown in Fig. 6. TCA API using PEACH3 achieves the largest bandwidth. It is about 1.21 times of that with PEACH2 and 1.54 times of that with MPI/Infiniband for 512KB data transfer. The maximum bandwidth is about 3.5GB/sec. However, the bandwidth of TCA APIs is degraded for data transfer more than 2MB, and MPI/Infiniband becomes better. The performance degradation of TCA APIs is caused by the increasing overhead of GPUDirectRDMA used in TCA APIs. MPI/Infiniband uses memory of host CPU for the data transfer more than 2MB to avoid the overhead.

4.4 Application Performance

4.4.1 The evaluation of BFS

As shown in the previous section, the traversed edges per second (TEPS) is a measure of the performance in Graph500, and *scale* shows the size of graph. First, we show the cases when COMM_FLAG, COMM_OUT and COMM_TREE are implemented with various combination of TCA API and CUDA API in Fig. 7.

In this application, since the data size of each communication is rather small, the case of using TCA API for all communications achieved the best performance with all scales. For large scale graphs, only using TCA API for COMM_OUT achieved the best performance, while for small

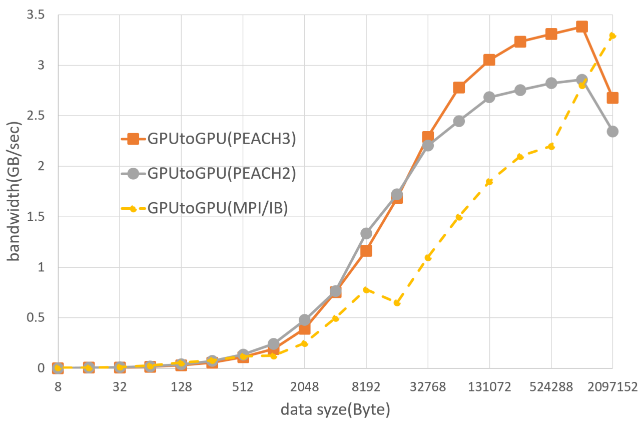

Figure 6: Bandwidth vs. Data size of inter-node communication

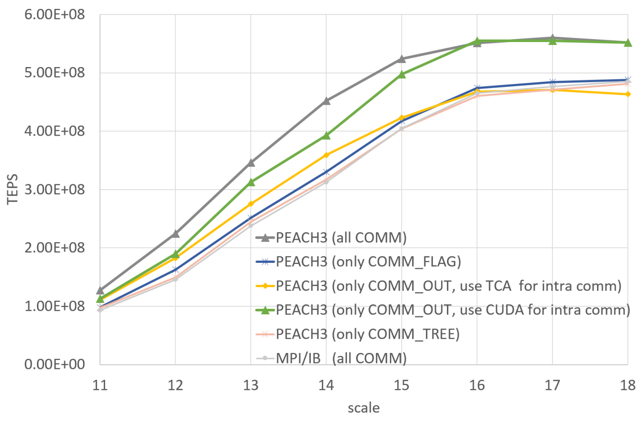

Figure 7: TEPS vs. scale when various combination of APIs is used

scale graphs, using TCA API for intra-node communication also improved the performance.

The performance comparison between the case with PEACH3, PEACH2 and MPI/Infiniband is shown in Fig. 8.

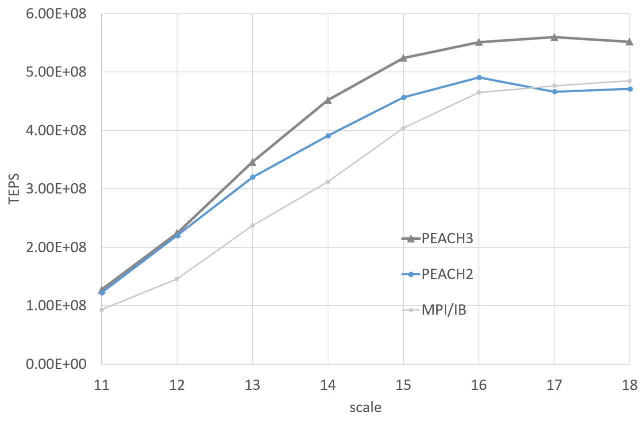

Figure 8: TEPS (PEACH3 vs PEACH2 vs MPI/IB)

For all scales, PEACH3 achieved the best performance. In the scale 12 and scale 13, the performance with PEACH3 and PEACH2 is almost the same because the minimum latency of PEACH3 is equal to PEACH2. When the scale is larger than 13, the performance of PEACH3 is better than PEACH2. It achieved about 1.16 times better than that with PEACH2, and 1.3 times better than that with MPI/Infiniband for *scale* = 15 graph. For large sized graph, the performance with PEACH2 is lower than that with MPI/Infiniband because of the lack of bandwidth, while PEACH3 with better bandwidth keeps the difference.

4.4.2 *The evaluation of CG*

In this evaluation, the implementation of the previous research[15] is almost directly used. We show the elapsed time of executing CG normalized to the case with MPI/Infiniband in Fig. 9. For CLASS=S with 1400 elements, the performance with PEACH3 is about 12% better than that with PEACH2 and 25% with MPI/Infiniband. Both COMM_EXCH and COMM_SPMV are reduced by using TCA API with PEACH3. Note that COMM_DOT is communication between CPUs and so MPI/Infiniband is used in all cases. However, since the bandwidth of PEACH3 with PCI gen3x8 is smaller than that of Infiniband with PCI gen3x16, the performance benefit was disappeared for CLASS=A matrix with 14000 elements and CLASS=B matrix with 75000. Strange to say, for CLASS A and B, the execution time of SPMV computing (not communication) with PEACH3 is larger than the others. It comes from the problem of GPU driver used in the system with PEACH3.

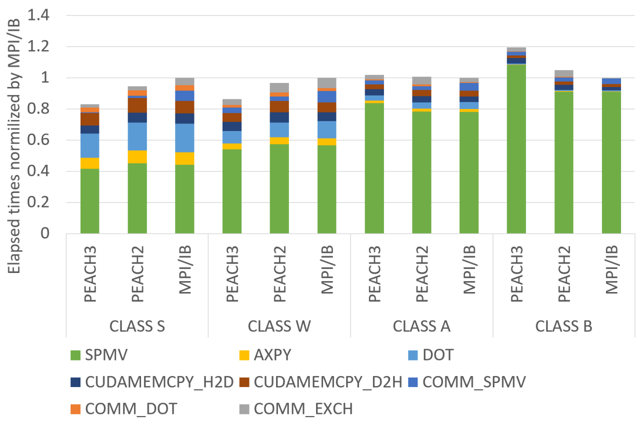

Figure 9: Elapsed time of CG (normalized the case with MPI/IB)

For large data size, TCA API spent larger time than that of MPI/Infiniband for COMM_EXCH. In such a case, CUDA API should be used for intra-node communication, and TCA API is used only for inter-node communication. Fig. 10 shows the communication time when such a usage is applied for CLASS B and C matrices. The total communication time can be reduced with PEACH3 even for CLASS C matrix for which the time with PEACH2 is worse than that of MPI/Infiniband.

4.4.3 *Guideline of API usage*

From the evaluation results, it appeared that PEACH3

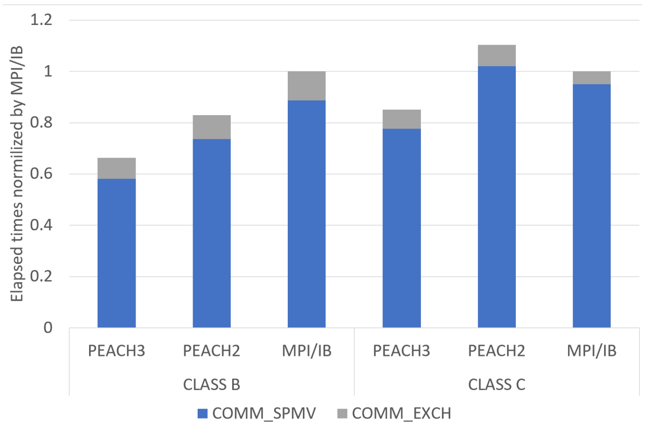

Figure 10: Communication time of CLASS=B and C (normalized the case with MPI/IB)

improved the performance of inter-node communication, when the exchanged data size is smaller than 1MB. As GPUDirect becomes the bottleneck with the current setting, performance will be improved by using MPI/IB if data size exceeds 1MB. BFS shows that PEACH3 was advantageous even for intra-node communication because of its low latency. However, CUDA API using PCIe Genx16 is superior to PEACH3 with PCIe Gen3 x8 when the data size exceeds 16KB. Programmers can improve their application performance by using appropriate API according to the exchanged data size.

5. CONCLUSION

An FPGA switching hub for tightly coupled accelerators architecture called PEACH3 was evaluated with application programs. Two application programs: BFS (breath first search) and CG (conjugate gradient) were implemented with TCA IP and CUDA IP with MPI/Infiniband. The performance of BFS with PEACH3 was 1.16 times better than that with PEACH2, and 1.3 times better than that with MPI/Infiniband for a graph with $scale = 15$. In CG, for the small matrix (CLASS=S), the PEACH3 achieved 12% better performance than that with PEACH2 and 25% with MPI/Infiniband. However, since the bandwidth of PEACH3 with PCI Gen3x8 is smaller than Infiniband with PCI gen3x16, the performance benefit was disappeared for CLASS=A matrix. Through the evaluation, it appears that if the data size is small, using TCA API with PEACH3 is advantageous even for intra-node communication.

HA-PACS provides both TCA and Infiniband. For such a machine, TCA and Infiniband can be switched as the size of data. If the data size is more than 1MB, Infiniband should be used. Otherwise, using PEACH3 is advantageous. Also, for intra-communication, if the data size is less than 16KB, TCA API should be used rather than CUDA API. The automatic switching mechanism of API is our future challenge.

ACKNOWLEDGMENT

The present study was supported in part 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”.

6. REFERENCES

[1] Global Scientific Information and Computing Center, Tokyo Institute of Technology, “What is TSUBAME?” http://www.gsic.titech.ac.jp/en/tsubame.
[2] OAK RIDGE National Laboratory, “Introducing Titan,” https://www.olcf.ornl.gov/titan/.
[3] S. Otani, H.Kondo, T. Hanawa, S. Miura, and T. Boku, “Peach: A multicore communication system on chip with pci express,” in *IEEE Micro*, 2011, pp. 39–50.
[4] T. Hanawa, Y. Kodama, T. Boku, and M. Sato, “Tightly coupled accelerators architecture for minimizing communciation latency among accelerators,” in *IEEE 27th IPDPSW*, 2013, pp. 1030–1039.
[5] U. of Tsukuba, “Supercomputers,” https://www.ccs.tsukuba.ac.jp/eng/supercomputers/#HA-PACS.
[6] T. Kuhara, T. Kaneda, H. Amano, T. Hanawa, Y. Kodama, and T. Boku, “A preliminarily evaluation of peach3: A switching hub for tightly coupled accelerators,” in *Proc. of the International Symposium on Computing and Networking (CANDAR)*, 2014, pp. 377–381.
[7] T. Kaneda, C. Tsuruta, T. Hanawa, and H. Amano, “Performance evaluation of peach3: Field programmable gate array switch for tightly coupled accelerators,” in *International symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, ser. HEART’16, July 2016.
[8] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel, “CAPI: A Coherent Accelerator Processor Interface,” in *IBM Journal of Research and Development*, 2015, pp. 1–7.
[9] NVIDIA, “NVIDIA NVlink High-Speed Interconnect: Application Performance,” NVIDIA, White Paper, 2014.
[10] R.Ammendola et el, “APEnet+: high bandwidth 3D torus direct network for PetaFlops scale commodigy clusters,” *Journal of Physics, ser. Conference Series*, vol. 331, Part 5, No.5, 2011.
[11] T. Kuhara, C. Tsuruta, T. Hanawa, and H. Amano, “Reduction Calculation in an FPGA based switching hub for high performance clusters,” in *FPL15*, Sept. 2015.
[12] C. Tsuruta, Y. Miki, T. Kuhara, M. Umemura, and H. Amano, “Off-loading LET generation to PEACH2: A switching hub for high performance GPU clusters,” in *International symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, ser. HEART’15, May 2015, pp. 1–6.
[13] “Graph 500,” http://www.graph500.org/.
[14] “Nas parallel benchmarks,” https://www.nas.nasa.gov/publications/npb.html.
[15] K. Matsumoto, N. Fujita, T. Hanawa, and T. Boku, “Implementation and Performance Evaluation of NAS Parallel CG Benchmark on GPU Cluster with Proprietary Interconnect TCA,” in *12th International Meeting on High Performance Computing for Computational Science (VECPAR2016)*, 2016.