Off-loading LET generation to PEACH2 : A switching hub for high performance GPU clusters

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

Yohei Miki
Center for Computational Sciences
University of Tsukuba,
1-1-1 Tennodai, Tsukuba,
Ibaraki 305-8577, Japan
ymiki@ccs.tsukuba.ac.jp

Takuya Kuhara
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
kuhataku@am.ics.keio.ac.jp

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

Masayuki Umemura
Center for Computational Sciences
University of Tsukuba,
1-1-1 Tennodai, Tsukuba,
Ibaraki 305-8577, Japan
umemura@ccs.tsukuba.ac.jp

ABSTRACT

A hardware local essential tree (LET) generator used in an N-body simulation is implemented on the FPGA of PEACH2 (PCI Express Adaptive Communication Hub ver2), a low latency switching hub for high performance GPU clusters. By using the pipelined on-the-fly execution with a multipole acceptance criterion judging module and a data updating module, the generation performance is 2.2 times faster than that with the CPU. When data communication is considered, the performance was 7.2 times as the case with the CPU.

1. INTRODUCTION

The field-programmable gate array (FPGA)-based switching hub is commonly used in various layers of networking. One such switching hub, PEACH2 (PCI Express Adaptive Communication Hub Ver.2), has been developed for low latency direct communication between accelerators through a PCIe standard I/O bus based on the concept of tightly coupled accelerators (TCA) architecture [2][7]. By using four PCIe ports provided to PEACH2, a double-ring network is formed between multiple nodes consisting of a host CPU and GPUs. Memories of the host CPU and GPUs attached to connected nodes are mapped onto a single address space of the PCIe and data can be transferred by writing it to this address. Hardwired logic of the FPGA in the PEACH2 chip is used for the packet buffer, switch, routing function, and DMA controller, just like any other switching hub. However, there is room to implement other functions specialized for applications executed in the system. As with other high performance computers, the main players with this computing are GPUs and CPUs connected to PEACH2, but the reconfigurable functions implemented with the hardwired logic in PEACH2 can additionally be used as another accelerator that helps compute the GPUs and CPUs. This approach is especially advantageous when the processed data need to be exchanged between GPUs and transferred through PEACH2. In other words, on-the-fly processing during data transfer is possible.

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2015) Boston, MA, USA, June 1-2, 2015.

To investigate this approach, here we implement a function to generate a locally essential tree (LET) in an N-body simulation that is commonly used in astrophysics [14]. This tree data structure was introduced so as to increase the number of particles to be simulated with a limited computational cost. When running a parallel tree method with multiple GPUs, the data of the particles are divided and distributed among each GPU, so locally divided trees must be exchanged between GPUs during the simulation. The LET is a tree structure proposed to reduce the amount of data that has to be exchanged between GPUs. In the conventional implementation, the LET is generated in the CPU and/or GPU depending on the data from a GPU and is then transferred to the different GPUs. By implementing the LET generator in the PEACH2 FPGA, the data can be directly transferred from one GPU to another via on-the fly processing for LET generation. This off-loading reduces the total amount of data transfer as well as the CPU and/or GPU load. It also concentrates the GPU in a process called Tree Walk.

The rest of this paper's organized as follows.

First, we explain TCA and PEACH2 in Section 2. Then, the algorithm for generating LET is introduced in Section 3. We describe our implementation of the hardware accelerator in PEACH2 in Section 4. We evaluate the performance improvement in Section 5 and then conclude with a brief summary in Section 6.

2. TCA/PEACH2

2.1 HA-PACS/TCA

HA-PACS/TCA (Highly Accelerated Parallel Advanced system for Computational Sciences / TCA) is a system de-

Table 1: HA-PACS/TCA system.

CPU	Intel Xeon-E5 2680v2 2.8 GHz 2 sockets IvyBridge-EP 10 core/socket
Memory	DDR3 1866 MHz 4 ch 128 GBytes
MotherBoard	superMicro X9DRG-QF
GPU	NVIDIA K20X 4
Memory(GPU)	GDDR5 6 GBytes/GPU
TCA board	Stratix IV EP4SGX530NF45C2
the number of nodes	64

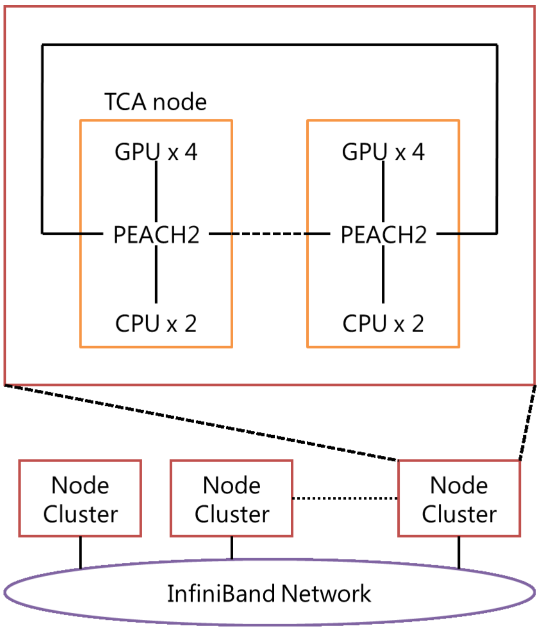

Figure 1: Overview of HA-PACS/TCA design.

veloped for removing communication bottleneck by using direct PCIe packet transfer between GPUs across nodes [7][2]. When we communicate between GPUs across nodes in multi-GPU clusters, three steps are needed. First, we copy the data from the GPU (node A) memory to the host CPU (node A) memory. Second, the data are transferred from the host CPU (node A) to another host CPU (node B). Finally, the data are copied from the host CPU (node B) memory to the GPU (node B) memory. Communication between the CPU and GPU, and between CPUs across multiple nodes, requires a large latency. This can sometimes result in a serious performance bottleneck. TCA architecture provides an FPGA board called PEACH2 (PCI Express Adaptive communication Hub ver2) in each node. TCA enables direct communication between nodes through PCIe, which is commonly used as a bus in a node. That is, communication between GPUs can be done through PEACH2 without having to pass through any host CPUs.

The specifications of HA-PACS/TCA are listed in Table 1. Each node has its own PEACH2, two Ivy Bridge CPUs, and four GPUs. Figure 1 shows an overview of TCA. Eight nodes are connected with a ring network formed with PEACH2. Two rings can also be connected with an extra link provided in each PEACH2. For connecting to a greater number of nodes, a higher-level Infiniband network is used.

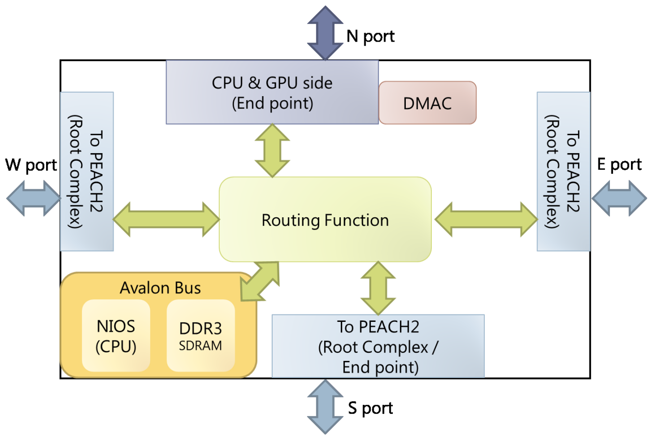

Figure 2: Block diagram of PEACH.

2.2 PEACH2

Figure 2 shows the block diagram of a PEACH2 chip implemented on Altera's Stratix IV FPGA [17]. The main role of PEACH2 is extending PCIe, which is commonly used only as an I/O network, to connect multiple nodes of a cluster. PEACH2 has four ports: N, E, W, and S. Port N, an endpoint for PCIe Gen2 x8, is pushed into the PCIe connector on the host CPU board. Port E, an endpoint Gen2 x8, and Port W, a root complex Gen2 x8, are used for interconnection between nodes. Port S is a selectable PCIe Gen2 x16 port and used to connect two ring networks formed by Ports W and E. The routing function embedded in the FPGA decides the destination port simply by checking the destination address of the PCIe packet on a single 512-GByte shared address space. The routing function provided in PEACH2 has the control registers for the address mask and the lower and upper bounds. The destination port is statically decided by checking the address with the address mask. On the PEACH2, a memory access to a remote node is restricted to a memory write request. Instead of memory read, which is difficult to implement efficiently, the proxy write mechanism can achieve the same effect by using driver support. DMAC supports sophisticated block data transfer in the address space.

Table 2 shows the details of PEACH2. It was implemented with Altera's Stratix-IV and works at a 250-MHz system clock. 512-MByte DDR3 SDRAM is provided on the board. The logic utilization is just 22%. Softcore CPU NIOS working at a 150-MHz clock is provided for the management of the PEACH chip. Figure 3 shows a photo of the PEACH2 board. Port N is implemented as a card edge while cable connectors are used for other ports. A daughter board is used to extend Port S to x16 port.

3. TARGET APPLICATION

3.1 Tree method for N-body simulation

N-body simulation is commonly used to investigate the structure formation and evolution history of galaxies by solving the following equation:

$$\boldsymbol{a}_i = \sum_{j=0 j \neq i}^{N-1} \frac{G m_j (\boldsymbol{x}_j - \boldsymbol{x}_i)}{(|\boldsymbol{x}_j - \boldsymbol{x}_i|^2 + \epsilon^2)^{\frac{3}{2}}}$$

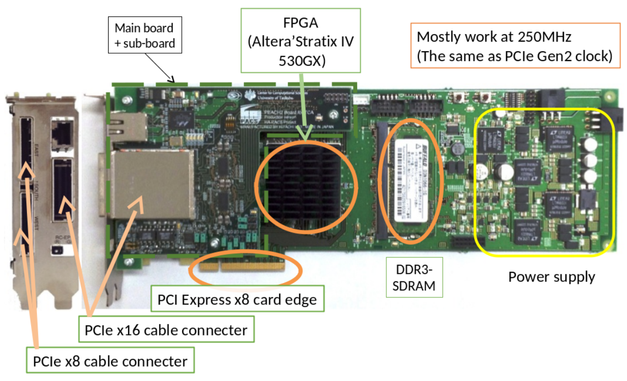

Figure 3: Photograph of PEACH2.

Table 2: Details of PEACH2.

FPGA Board	Stratix IV EP4SGX530NF45C2
Number of pins	1932 (GND : 497, VCC : 298)
User pins	379 / 1112 (34%)
Logic Utilization	22%
Combinatinal ALUTs	63930 / 424960 (15%)
Dedicated logic registers	78236 / 424960 (18%)
Total block memory bits	2947383 / 21233664 (14%)
DSP block elements	4 / 1024 (1 %)
NIOS II	150 MHz
Total GXB RX PCS	32/ 32 (100%)
Total GXB TX PCS	32/ 32 (100%)

where $\boldsymbol{x}_i$ is a coordinate, m_i is a mass, and $\boldsymbol{a}_i$ is an acceleration of the i-th particle out of N particles. G is the gravitational constant and ϵ is the gravitational softening length introduced to avoid divergence due to division by zero. The simplest N-body simulation is a direct summing up of all gravity from $(N-1)$ particles. However, it takes O$[N^2]$ computational cost, which makes it difficult to treat a large N.

The tree method has been proposed to reduce the computational cost to O$[Nlog(N)]$ [11] by using the multipole expansion technique. If the position of node j is far enough away from particle i, no further search is required. This reduces the amount of calculation needed. In this process, to judge the distance between nodes, we adopt Multipole Acceptance Criterion (MAC) [15][16].

Figure 4 gives a breakdown of the execution time for each step of the N-body simulation when it is executed in a single node including a CPU and GPUs. The "walk tree" is the time it takes to follow up tree data, "make tree" is the time spent constructing the tree, "calc MAC" is the time to calculate the position, mass, and MAC of all tree nodes, "PH-key" is the time to generate the Peano-Hilbert key [8] of the N-body particles and sort them using the key, "body" is the process of transferring the particle data between a CPU and a GPU, and "node" is the process of transferring tree data to GPUs. In this measurement, the GPU we used was a Tesla K20 and the number of particles N was 2^{23}. Here, we divide the distribution of the "walk tree" into 3 parts because we use a block time step approach. We also have to make a tree at appropriate intervals in every step, so "PH-key" and "make tree" have sparse intervals. Here, the gravitational calculation time (sum of "calc MAC" and

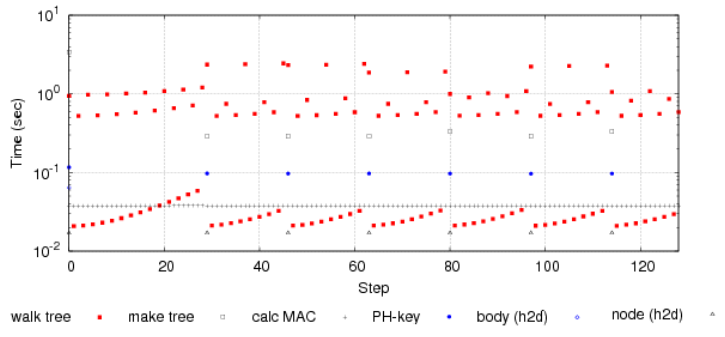

Figure 4: Breakdown of execution time.

"walk tree") must be dominant. That is, "body" and "node" for data transfer must be smaller than the calculation time. It is for these reasons that we decided to use LET and make LET generater.

3.2 Locally Essential Tree (LET)

When a tree method is executed in a GPU cluster such as HA-PACS, the tree is divided and distributed to each GPU for parallel processing. A locally essential tree (LET) [14][6][10][9] is a tree with a pruned data structure that reduces the communication between GPUs. In order to generate a LET, first, a GPU I sends the imaginary particle data to GPU J. This particle is the sphere including all particles in GPU I. The center of the sphere is consistent with the focus of particle distribution. GPU J performs tree traversal and judges on the basis of MAC whether a node needs to be added to the tree and then sends the node to GPU I if needed. GPU I receives the node and adds it to the LET. The same process is performed for all combinations of GPUs in the system to form a LET in each GPU.

3.2.1 Tree data structure

Here, we adopt a tree data structure optimized for BFS. We used five arrays, as shown in Figure 5.

- *list[leaf_level]*: Only this array is indexed by (leaf_level), which represents the depth of the tree (e.g., if the root node has 0 and its children nodes have 1). The array of the list shows the index of each starting leaf_level and the number had by leaf_level. Both items are represented with a 32-bit integer, so 64 bits are needed in each element.

- *jpos[id]*: The rest of the arrays are indexed by the node identifier (id). The array *jpos* contains the radius, x coordinates, y coordinates, and z coordinates of the node. Each data item is represented by a single-precision floating point number, so 128 bits are used in total for each node.

- *mj[id]*: It shows the mass per node with a single-precision floating point, so 32 bits are needed.

- *mask[id]*: The element of this array takes a value of 1 or 0 to show whether input node data is included in the output LET data. If the *mask[id]* is 1, the data of its id must be included in the LET, otherwise it is not included. At the initial state, only the root node has 1.

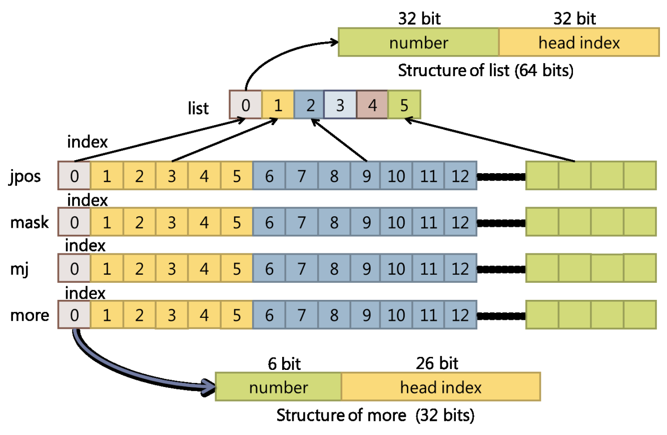

Figure 5: Structure of array.

- *more[id]*: This array shows information of the child nodes. In this implementation, the upper 6 bits hold the number of child nodes and the lower 26 bits hold the head index of the child nodes. For example, when the upper 6 bits are 3 and the lower 26 bits represent 10, children nodes are node 10, 11, and 12.

3.2.2 *Flow of LET creation*

Figure 6 shows the flow of LET creation. First, we check the array of *mask[id]*. When *mask[id]* is 1, the next step is to judge distance. Otherwise, some part of the particle data is changed and moved to the next node. The judge distance step is for comparing distance by using MAC. Next, after searching *more* arrays, the *mask* array of a child node is changed from 0 to 1. Some of the interior of particle data is then updated and added to the output LET. After these processes are finished, we move on to the next node. The checking is performed in the BFS order.

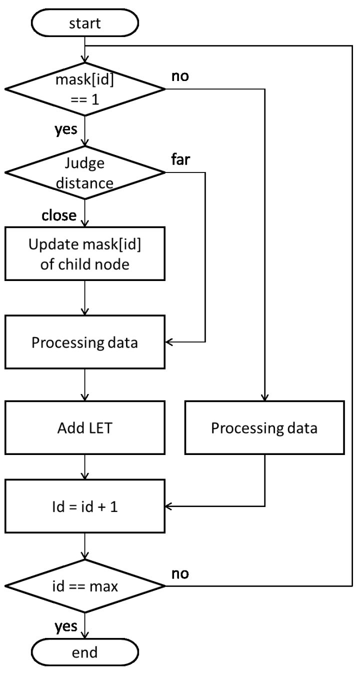

Figure 6: Flow of LET creation.

4. ON-THE-FLY LET GENERATOR

4.1 Implementation in PEACH2

In our implementation, the LET generation is done in the host CPU communicating with GPUs. However, in TCA, GPUs can communicate directly with PEACH2. Implementing the LET generator in PEACH2 has the following benefits:

- The data to generate LET must be transferred between GPUs through PEACH2, and LET generation can be done on-the-fly during data transfer.

- Data transfer between CPU and GPU for generating LET is removed. This saves communication time as well as computational load on the CPU and/or GPU.

In PEACH2, the main functions as switching hub include PCIe interface, routing functions, multiplexers for packet switching, and the DMA controller working at a 250-MHz clock. An LET generator must be implemented so as not to influence the operation of this part, so we implemented our logic as a functional module attached to the Avalon bus, as shown in Figure 2. The working memories inside the FPGA, DDR3 DRAM interface, and Nios soft processor are connected to the bus, and this part works at a 150-MHz clock for easier implementation. The memory can be mapped into the same address space as the GPU and CPU memory modules connected with PEACH2, and writing this memory area is done by sending data to the registers of the LET generator. Since the bus width is 128 bits, the fundamental data size in the implementation is set to be 128-bit.

4.2 The LET generator

As shown in Figures 7 and 8, the LET generator consists of a MAC judging module and a data updating module. Input data structures *jpos*, *more*, and *mj* come from a PCIe link on the 128-bit data bus with 2 clock cycles to the MAC judging module, which is shown in Figure 9. In order to receive data continuously, all computational modules are executed in a pipelined manner. Altera's floating point megafunctions are used for the single-precision floating point calculation. All megafunctions are pipelined so that the MAC judging module can accept the input data in every clock cycle. However, because of the limitation of the 128-bit internal bus, the module accepts and generates data once every two clock cycles. The latency of the module is 136 clock cycles.

If the result of judging a module is ”1”, the data updating module updates the data in *more* and *mask* and the *mask* of the children nodes must be set to ”1”. Since the tree is pruned, *more* must be modified in accordance with the number of pruned nodes. Note that most of the processing in the LET generator can be done without locking pipeline, expect when the index crosses the leaf_level. An *mask* needs to be updated if its children nodes have a leaf_level one greater than its own. Thus, the processing must be stalled until the *mask* data have updated at the pipeline by 140 cycles at most.

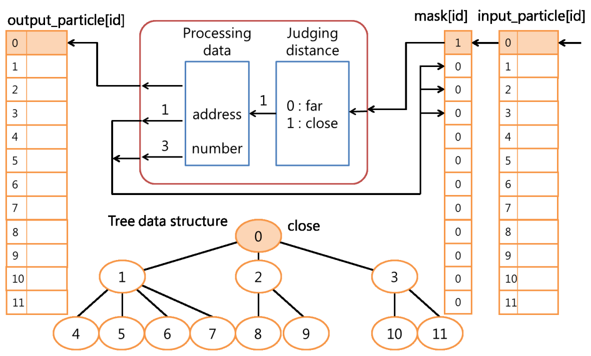

Figure 7: LET generator (id = 0).

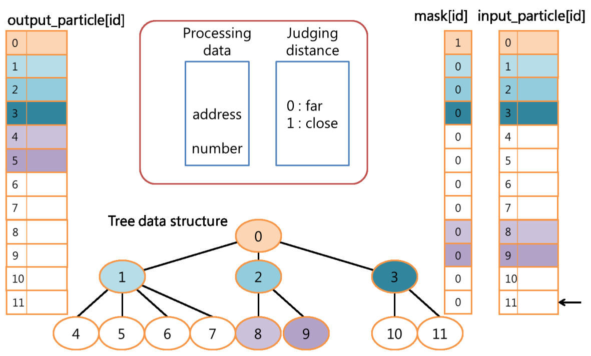

Figure 8: LET generator (id = 11).

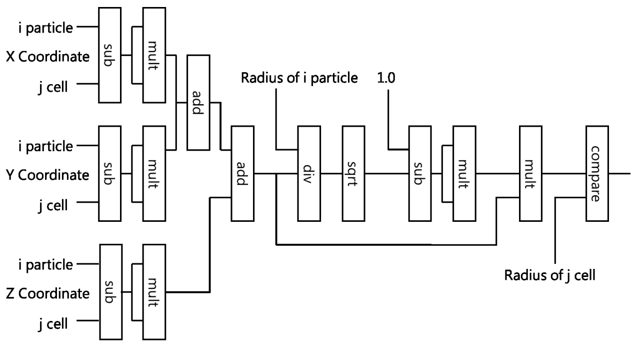

Figure 9: The MAC judging module.

5. EVALUATION

In this section, we evaluate our implementation of the LET generators in PEACH2 and compare it with a case in which an LET is generated in a host CPU. Table 3 lists the details of the evaluation environment. The LET generator was designed in Verilog HDL and synthesized with QuartusII 13.1. As shown in Table 4, the hardware used is small enough to be implemented in PEACH2 without disrupting the main switching functions and our design enables fast enough operation to be implemented on the Avalon 150-MHz bus.

Table 5 shows the execution time of the LET generator and CPU. The clock cycles in the LET generator are counted using RTL simulation and the execution time of the CPU is the average of 10 executions. The difference in time scale between Table 5 and Table 4 (breakdown of execution time) can be attributed to particle number. It appears that the LET generator works 2.2 times faster than the CPU. The speed of the LET generator was mainly limited by the rate of the particle data thrown into the module once every 2 clock cycles and the pipeline stall for updating *mask*.

The above results only pertain to the computation time for LET generation, not the communication time. In the N-body simulation that are underway currently implemented, the data has to be transferred from the GPUs to the CPU and then the result has to be transferred back to the GPUs again. In contrast, in the PEACH2 implementation, the computation time of the LET generator is completely overlapped with the data transfer since it is on-the-fly computation. Table 6 shows the execution times including the data communication. The hardware excution time is the sum of LET generator and the time of the communication delay in the past. The CPU execution time is an average of 10 executions, as well as the execution time of only LET gengeration. As shown in this table, the LET generator in PEACH2 achieves a 7.2 times faster execution than the CPU.

Table 3: Evaluation environment.

FPGA	Stratix IV EP4SGX530NF45C2
Language	Verilog-HDL
Tool	Quartus II 13.1
CPU	Intel Xeon E5 2680
ICC	icc 13.1.3
MPI	Open MPI 1.6.5
CUDA	CUDA 5.5
the number of particles	4096

Table 4: Hardware used.

Logic Utilization	12 % 34 % (add PEACH2)
Combinatorial ALUTs	15692 / 424960 (4 %) 79622 (19 %) (add PEACH2)
Dedicated logic registers	36623 / 424960 (9 %) 114859 (27 %) (add PEACH2)
Total block memory bits	4608 / 21233664 (1 %) 2951991 (14 %) (add PEACH2)
DSP block elements	36 / 1024 (4 %) 40 (4 %) (add PEACH2)

Table 5: Execution time to construct LET.

	execution time
CPU	69.3 μsec
Our off-loading module	31.4 μsec

Table 6: Execution time including communication.

	Execution time
CPU	245.7 μsec
Our off-loading module	33.7 μsec

6. RELATED WORK

On-the-fly acceleration in the FPGA switching hub or network interface has been tried mostly for networking. Encryption and decryption are practical targets of the acceleration, and there has been a lot of research on implementing hardware mainly on a network interface FPGA [12]. One common technique is accelerating the network protocol to reduce the overhead of CPU with hardware implemented on a network interface FPGA. Reducing the time it takes to compute CRC is one of the targets of such acceleration [3][1].

Recently, for big data processing, some of the functions for database processing have been off-loaded to network interface FPGAs [5]. Most of them are off-loaded to a network interface FPGA, and the target application is networking or database processing. To the best of our knowledge, there has been no research on off-loading part of a scientific computation into a switching hub FPGA. This is due to the difficulty of implementing additional hard-wired logic on the high speed switching hub used in supercomputers. Fortunately, although PEACH2 is a low latency switching hub for GPU clusters, it was designed so that extra functional modules can be attached by using the hardware resources that are not being used for the main function.

There are many accelerators and dedicated machines for N-body simulation, including various GRAPE models [13][4]. However, we can omit these here, since the focus of this research is not this type of dedicated accelerator.

7. CONCLUSION

An on-the-fly LET generator is proposed and implemented on PEACH2, a switching hub for high performance GPU clusters. Using the pipelined on-the-fly execution with a MAC judging module and a data updating module resulted in a generation performance 2.2 times faster than with a CPU. When data communication is considered as well, a performance 7.2 times faster than using the CPU can be achieved.

The total execution of N-body simulation using the proposed LET generator is currently under development. In future work, we will evaluate the effect of this generator on total N-body simulation with the HA-PACS/TCA system.

Acknowledgment

The present study was supported by the JST/ CREST program entitled " Research and Development on Unified Environment of Accelerated Computing and Inter- connection for Post-Petascale Era " in the research area of " Development of System Software Technologies for post- Peta Scale High Performance Computing. "

8. REFERENCES

[1] A.Alagic and H.Amano. Performance analyssi of fully-adaptable CRC accelrators on an FPGA. In *Proceedings of the International Conference on Field Programmable Logic and Application (FPL'12)*, pages 575–578, Sept 2012.

[2] U. o. T. Center for Computational Sciences. http;//www.ccs.tsukuba.ac.jp/.

[3] C.Toal, K.McLaughlin, S.Sezer, and X.Yang. Design and Implementation of a Field Programmable CRC Circuit Architecture. In *IEEE Trans. on VLSI Systems, Vol.17(8)*, pages 1142–1147, Aug. 2009.

[4] Ebisuzaki, Toshikazu ; Fukushige, T. ; Taiji, Makoto ; Makino, J. ; Sugimoto, D. ; Ito, T. ; Okumura, S.K. ; Hashimoto, E. ; Tomida, K. ; Miyakawa, N. GRAPE: special purpose computer for simulations of many-body systems . In *Proceeding of the Twenty-Sixth Hawaii International Conference onSystem Sciences*, pages 134–143, Jan 1993.

[5] E.S.Fukuda, H.Inoue, T.Takenaka, D.Kim, T.Sadahira, T.Asai, and M.Motomura. Caching Memcached at Reconfigurable Network Interface . In *Proceedings of the International Conference on Field Programmable Logic and Application (FPL'14)*, Sept 2014.

[6] Go Ogiya, Yohei Miki, Taisuki Boku, Masao Mori, Naohito Nakasato. Implementation and Performance Evaluation of Astrophysical Tree-code for GPU Clusters. In *Information Processing Society of Japan Vol.6*, pages 58–70, April 2013.

[7] T. Hanawa, Y. Kodama, T. Boku, and M. Sato. Interconnect for tightly coupled accelerators architecture, "IEEE 21st Annual Sympsium on High-Performance Interconnects (HOT Interconnects 21)", 2013.

[8] Hans Sagan. *Space-Filling Curves*. Springer-Verlag Berlin and Heildelberg GmBH and Co. K, 1994.

[9] J.Bedorf, E.Caburov, M.S.Fujii, K.Nitadori, T.Ishiyama, S.Portegies Zwart. 24.77 Pflops on a Gravitational Tree-Code to Simulate the Milky Way Galaxy with 18600 GPUs. In *Proceedings of the International Conference for High Performancs Computing, Networking, Strage and Analysis*, Dec 2014.

[10] J.Bedorf, E.Gaburov, S.Portegies Zwart. A sparse octree gravitational N-body code that runs entirely on the GPU processor. In *Journal of Computatinal Physics, Vol 231*, pages 2825–2839, April 2012.

[11] Josh Barnes, Piet Hut. A hierarchical O($NlogN$) force-calculation algorithm. In *Nature*, pages 446–449, December 1986.

[12] J.Szefer, Y.Chen, and R.B.Lee. General Purpose FPGA platform for efficient encryption and hashing . In *IEEE International Conference on Application spcific Ssytem Architectures and Processors*, pages 309–312, June 2010.

[13] Makino, J. ; Ito, T. ; Ebisuzaki, Toshikazu ; Sugimoto, D. GRAPE: a special-purpose computer for N-body problems. In *Proceedings of the International Conference on Application Specific Array Processors*, pages 180–189, Sep 1990.

[14] Michael S. Warren, John K. Salmon. Astrophysical N-body Simulations Using Hierarchical Tree Data Structures. In *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, pages 570–577, Sep 1992.

[15] Michael S. Warren, John K. Salmon. A Parallel Hashed Oct-Tree N-Body Algorithm. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, pages 12–24, March 1993.

[16] Michael S. Warren, John K. Salmon. Skeletons from the treecode closet. In *Journal of Computational Physics (ISSN 0021-9991) vol.111, No 1*, pages 136–155, March 1994.

[17] Yuetsu Kodama, Toshihiro Hanawa, Taisuke Boku, Mitsuhisa Sato. PEACH2: An FPGA-based PCIe network device for Tightly Coupled Accelerators. In *HEART2014*, June 2014.