Performance Evaluation of a CPU-FPGA Hybrid Cluster Platform Prototype

Yasunori Osana and Yohei Sakamoto
University of the Ryukyus
1 Senbaru, Nishihara, Okinawa 903-0213 Japan
{osana,skmt}@lut.eee.u-ryukyu.ac.jp

ABSTRACT

In this paper, a CPU-FPGA hybrid cluster architecture and its prototype implementation is introduced. The prototype system consists of 2 CPUs and 3 FPGAs. One FPGA is connected to the host CPU via PCI Express and high-speed serial link connects between 2 FPGAs. Processing elements for stream computing can be loaded on these FPGAs by partial reconfiguration. The effective throughput of PCIe is 1.5GB/s in single duplex and 2.4GB/s in full duplex. Serial link has 780MB/s in single and 1.5GB/s in full duplex. With a simplified 2D diffusion equation example, stream processing on the FPGA achieved 41.95 GFLOPS. Also, the cluster's CPU-FPGA and FPGA-FPGA communcation mechanism is based on FIFO-based IP cores, so the cluster system itself and its application can be easily ported to other FPGAs.

1. INTRODUCTION

As the "power wall" comes obvious with the rising operating frequency of microprocessors, now accelerators are beginning to play an important role in data centers and high-performance computing. GPGPUs are already widely used in many supercomputers in TOP500, attracting many users in large-scale applications such as scientific computing and machine learning.

With the improvements of high-level synthesis technology, FPGAs are also becoming popular as custom hardware accelerator device. Some commercial data centers[1] or cloud platform[2][3] are already in service with FPGAs. Of course, there are also multi-FPGA systems in research field[4][5][6].

Most of these FPGA-enabled computing systems consist of multiple microprocessors and multiple FPGAs to exploit both of software and hardware parallelism. However, the application on these systems are not fully portable because these systems usually consist of custom-made FPGA boards, or dedicated CPU-FPGA / FPGA-FPGA interface.

To resolve this problem, several FPGA operating systems such as LEAP[7] had proposed, but still not widely used. Instead, this paper proposes a portable light-weight framework for CPU-FPGA hybrid cluster, consists of FIFO-based CPU-FPGA and FPGA-FPGA interfaces. Although small modifications and recompiling are required, the framework can be ported to virtually any Xilinx FPGA board with PCIe and SerDes interface.

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

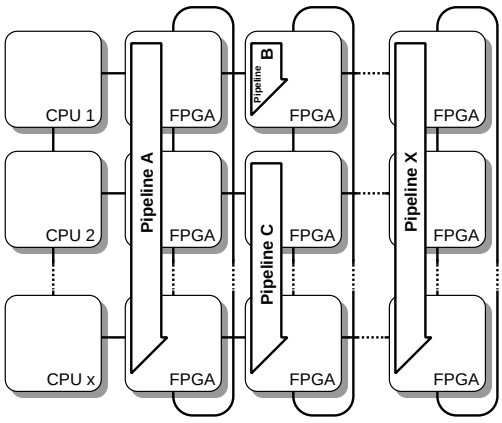


Figure 1: CPU-FPGA Hybrid Cluster Architecture

The main aim of this CPU-FPGA hybrid cluster framework is to accelerate stream oriented scientific computation. This paper shows its basic organization, prototype system, CPU-FPGA / FPGA-FPGA communication, partial reconfiguration based management mechanism and their basic performance evaluation.

2. CPU-FPGA HYBRID CLUSTER

2.1 Overview

Figure 1 shows the organization of the CPU-FPGA hybrid cluster. The CPU nodes are connected by Gigabit Ethernet, thus programmers can use MPI and any other parallel and distributed programming tools. Each CPU node have an FPGA board connected via PCI Express, and the PCIe-connected FPGAs have a dedicated ring network.

The FPGAs have several pipelined processing elements (PEs), and some of them may be spanning on multiple FPGAs using the ring network. These PEs are to offload several subroutines or kernels from the software on CPU nodes, and they can be accessed from any CPU node using the PCIe interface and the ring network. Also, multiple rings can be added by using the same dedicated network, as illustrated the in figure. Adding more rings enables to accommodate more PEs, or enables more subroutines to be offloaded.

Because all PEs are shared by all CPU nodes, MPI-parallel processes on CPU nodes can share their offloaded subroutines on the FPGA domain. This will also help to keep all PE pipelines busy, by aggregating requests from multiple

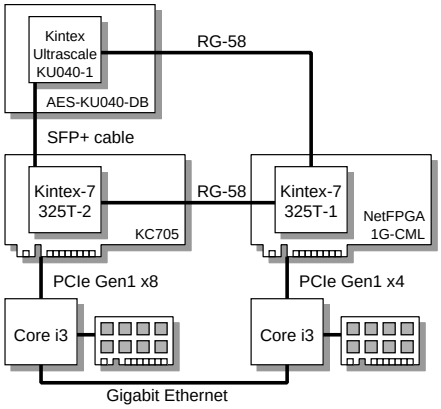


Figure 2: Organization of Prototype System

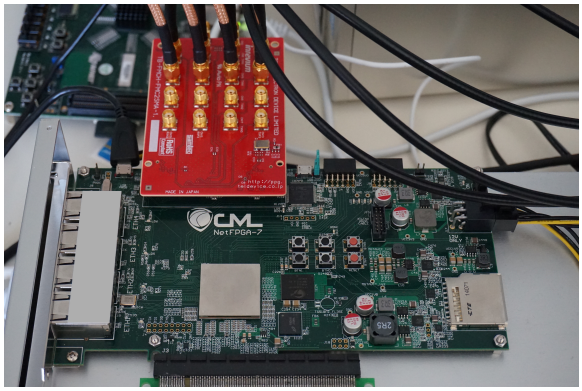


Figure 3: NetFPGA-1G-CML board with FMC-SMA card

CPU nodes.

PCIe and FPGA-FPGA network controllers are implemented on the FPGAs statically, and PEs are loaded as partial reconfiguration bitstreams. The communication and partial reconfiguration mechanisms are described in the following sections.

2.2 Prototype System

To confirm the effectiveness of the idea described in previous section, a prototype system consists of 1 CPU and 2 FPGAs is installed and working. As shown in Figure 2, 2 Intel Core i3 hosts with Kintex-7 FPGA boards and a Kintex Ultrascale FPGA board are connected together by high-speed serial links.

The Kintex-7 boards are attached to the hosts by PCI Express (PCIe). 2 different Kintex-7 boards are installed to separate board-specific part of the design: Xilinx KC705 and Digilent NetFPGA-1G-CML. As the result, RTL implementations for these boards are almost same except the PCIe / serial link IP core and its wrapper module.

Another FPGA board with Kintex Ultrascale has no PCIe interface, but just connected to the Kintex-7 boards via the high-speed serial links. The physical media of serial links are RG-58 coaxial cables or SFP passive copper cable, all 6.25Gbps full-duplex.

From the host-software viewpoint, the host-attached Kintex-7 FPGAs is the "Local FPGA", which has direct connection

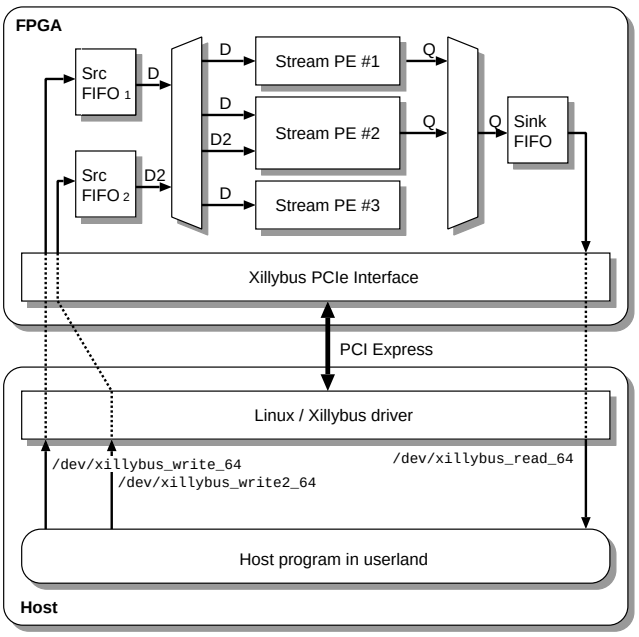


Figure 4: Host to Local FPGA Communication

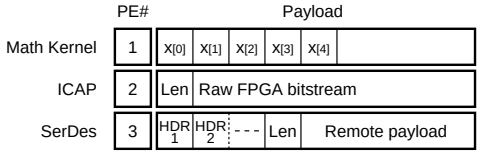


Figure 5: Data Frame Format to PEs

via PCIe. Other FPGAs are referred as "Remote FPGA." The Kintex-7 FPGAs can be both Local or Remote FPGA, determined by which host drive the FPGA.

One of the board, NetFPGA-1G-CML is shown in Figure 3. Because NetFPGA-1G-CML board has neither SFP or coaxial connector, an FMC-SMA daughter card is attached.

3. DESIGN AND IMPLEMENTATION

3.1 Host-FPGA Communication

Ubuntu Linux 16.04 is running on the host CPU, and the FPGA-to-Host interface is built on PCIe Xillybus[8] Rev.B core. Because Xillybus PCIe driver is included in the Linux kernel comes with Ubuntu, no modification is made on the system software.

PCIe Xillybus core is configured to have 2 write FIFOs and 1 read FIFO as shown in Figure 4. All FIFOs are 64-bit width and each has its own DMAC with the maximum data rate of 1600MB/s. In Linux, the FIFOs has separate device file entry under `/dev` directory, so host program in userland can use `read()` and `write()` function calls to access the FIFOs on the FPGA.

3.2 Stream PE and Partial Reconfiguration

As drawn in Figure 4, several stream PEs are placed between the Source FIFO (Rx on FPGA) and Sink FIFO (Tx on FPGA). PEs are basically a kind of arithmetic pipeline, but anything with FIFO interface is possible. For example,

controllers for FPGA-to-FPGA link and partial reconfiguration are implemented as stream PEs. Some PEs have only 1 input port (D) while others have 2 input (D and D2). Similarly, most PEs have 1 output port (Q) but some PEs have no output port.

Data stream is exchanged in "Frames" between the host program and the FIFOs. On local FPGA, a data frame to FPGA begins when the host program opens the FIFO device file, and ends when the program closes it. For data frame from FPGA, host program always know to know when the data transmission starts and ends (or frame length to receive), so host program can safely open and close the Sink FIFO device without losing any data fragment.

Because there are usually multiple PEs implemented on an FPGA, an "active" PE is selected by PE ID, the first 64bit word of a data frame. Figure 5 shows several examples of data frame format. Contents of second and rest words of a frame has PE specific format. For example, simple math kernel PE will receive the rest of frame as an array.

For example, the partial reconfiguration controller (ICAP PE) is implemented as one of these PEs, which accepts the second data word as the length of bitstream and the rest for configuration bitstream. The PE writes the configuration bitstream through ICAPE2 primitive, the internal configuration access port of FPGA. The PE has 64bit-input 32bit-output non symmetric FIFO internally because ICAPE2 accepts configuration bitstream just same as in 32bit Slave SelectMAP interface.

ICAP PE and ICAPE2 accepts a raw binary bitstream generated by Vivado design suite. Of course, the partial bitstream for partial reconfiguration must be prepared to be consistent with the preloaded static logic.

3.3 Remote FPGA Communication

To communicate with the remote FPGA, the FPGAs are connected by serial transceivers and cables, using Aurora 64b66b core[9] provided by Xilinx. The Aurora 64b66b core provides 64bit width AXI4-stream interfaces for both Tx and Rx, with data frame support.

SerDes PE is implemented to attach Aurora core Tx/Rx interface to the Stream PE interface. This PE works as a Xillybus-Aurora bridge on Local FPGA, then the same module provides Source and Sink FIFO for PEs on Remote FPGA. The FIFOs provided on the Remote FPGA acts just same as in Local FPGA, Remote FPGA can have multiple Stream PEs, too. Thus, Remote PE has other SerDes PEs to form FPGA-domain network.

As shown in Figure 5, the data frame begins with variable length routing header, to choose the remote PE and next destination of the datastream. A routing header word is required to go through an FPGA: the first header word is removed on each FPGA while the following header words just pass through. In this way, the datapath spanning multiple FPGAs are routed by the host.

3.4 Clock Domains and Flow Control

The Xillybus core with PCIe Gen1 x8 runs at 250MHz, ICAP PE runs at 100MHz due to the frequency limit of ICAPE2, and Aurora core at 6.25GHz has its Tx/Rx interface at 97.65625MHz (=6250/64.) To get these cores work together, asynchronous FIFOs are used across clock domains as shown in Figure 6. Because the Aurora clock is lower than 100MHz, everything on Remote FPGA is running with this

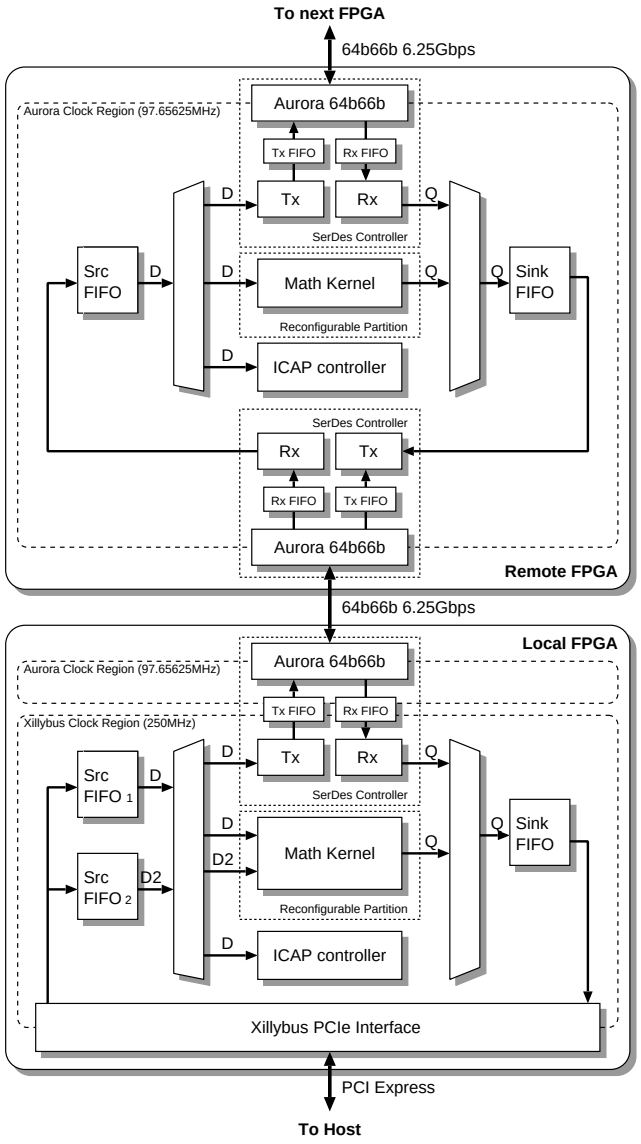
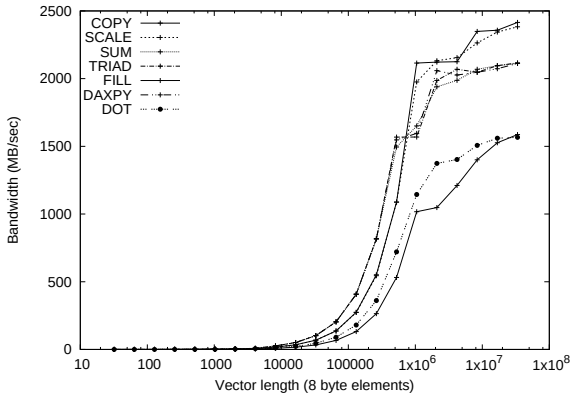


Figure 6: FPGA-to-FPGA Details

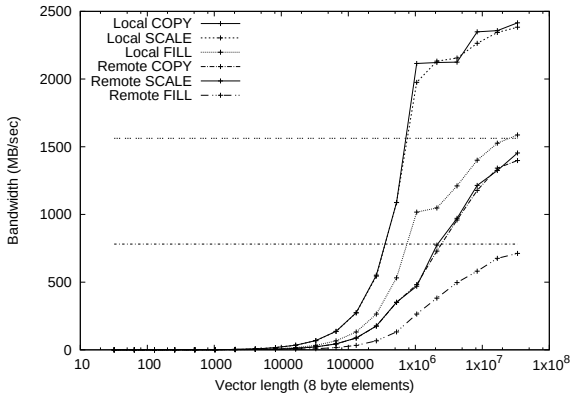
clock in current prototype. This will be changed in future implementation with faster link bitrate or larger PCIe bandwidth.

In addition to clock isolation, these FIFOs and Source / Sink FIFOs are also playing important role on flow control, to give proper back pressure to data source. Back pressure control is usually done by controlling "read enable" signal in upstream FIFO by "programmable full" signal of downstream FIFO. Xilinx FIFO core provides programmable full threshold programmability at runtime, so Stream PEs can control the threshold to fit their requirement.

In SerDes PE, programmable full signal of FIFO triggers to send flow control message, through NFC (native flow control) port of Aurora core. Also, Xillybus core recognizes the full signal of Source FIFO as back pressure control signal. By this mechanism, FIFO full can propagate beyond the serial link to prevent any FIFO overflow.



(a) All kernels on Local FPGA



(b) 1-input kernels on Local/Remote FPGA

Figure 7: STREAM/STREAM2 Results

4. EVALUATION

4.1 Basic Resource Usage

Everything described in the previous section is written in Verilog HDL, then implemented with Vivado Design Suite 2016.4. Both Local and Remote FPGA contains a partial reconfiguration partition to accommodate a dynamically-loaded stream PE.

Table 1 shows the resource usage breakdown, except the dynamically loaded PE. Xillybus core occupies a little bit larger area, but it's less than 4% of the total LUTs. Aurora (core + SerDes PE) consumes about 2000 LUTs, but this is still enough compact to build more FPGA-to-FPGA link ports on remote FPGA. There are enough room to implement Stream PEs for computation in all FPGAs.

4.2 Local and Remote Bandwidth

STREAM[10] and STREAM2[11] benchmarks are modified to measure Host-FPGA communication bandwidth. Both benchmarks evaluate bandwidth on floating-point vector operations, with fixed vector length (STREAM) or varying vector lengths (STREAM2.) The benchmarks contain following math kernels:

- STREAM:
  - COPY: a(i) = b(i)
  - SCALE: a(i) = q*b(i)
  - SUM: a(i) = b(i) + c(i)
  - TRIAD: a(i) = b(i) + q*c(i)
- STREAM2:
  - FILL: a(i) = q
  - DAXPY: a(i) = a(i) + q*b(i)
  - DOT: sum = sum + a(i) * b(i)

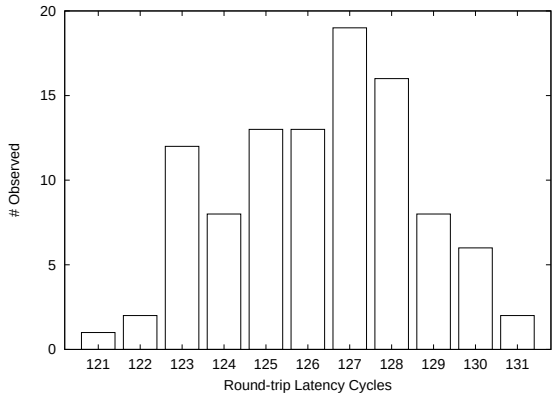


Figure 8: Round-trip Latency Cycles

In this work, STREAM2 code is modified to run these kernels (including STREAM kernels) on the FPGA, with varying vector length from 32 to 32×2^{20}. The kernels consist of several pipelined double precision floating-point cores from Xilinx, and loaded dynamically on the FPGA by partial reconfiguration through ICAP PE. The kernel implementation results are summarized in Table 2. All of these kernels work at 250MHz.

Figure 7 (a) shows the vector length and bandwidth on the Local FPGA. The bandwidth is low while the vector length is short, then goes saturating about 10^6 elements, or 8MB. FILL kernel and DOT kernel mostly use only 1 DMA channel, and their maximum bandwidth is about 1.5GB/s. This is close to 1.6GB/s, Xillybus's maximum DMA transfer rate, so DMA is working efficiently with enough long vectors.

Kernels work on 2 vectors are COPY and SCALE had saturated around 2.4GB/s, both reads 1 vector and writes 1 vector. Saturating bandwidth of kernels works on 3 vectors, SUM, TRIAD and DAXPY was about 2.1GB/s. From these results, it is clear that using many DMACs simultaneously doesn't always provide better performance.

Figure 7 (b) is the comparison of Local and Remote FPGAs with COPY, SCALE and FILL kernels. Other kernels are not still ported to Remote FPGA because only 1 source FIFO is implemented on Remote FPGA, but this will be fixed in near future.

Single input kernels in the list: COPY, SCALE and FILL were also implemented and tested on Remote FPGA as shown in Figure 7 (b). Horizontal dotted lines in the figure are 6.25Gbps and 12.5Gbps. Referring these lines, Remote FPGA bandwidth is saturating at just below the SerDes line rate: FILL kernel at single duplex rate (6.25Gbps), SCALE and FILL at full duplex rate (12.5Gbps).

4.3 Communication Latency

Round-trip latency cycles had measured with Vivado's ILA (internal logic analyzer) on the Local FPGA, and just

Table 1: Resource Usage Summary

	KC705			NetFPGA-1G-CML			KU040-DB		
Module	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM
Aurora	2279	3829	6	2219	3891	6	2004	3994	8
Xillybus	7433	6712	12	6065	4926	12	—	—	—
Slave I/F	—	—	—	—	—	—	332	170	0
Others	172	231	2	185	296	2	278	0	0
Total	9884	10772	20	8469	9113	20	2614	4164	8
Total %	4.8	2.6	4.5	4.2	2.2	4.5	1.1	0.9	1.3

Table 2: Resource Usage in Reconfig. Partition

Kernel	LUT	FF	BRAM	DSP
COPY	66	65	0	0
SCALE	368	893	0	10
SUM	679	1342	0	3
TRIAD	1012	2234	0	13
FILL	241	130	0	0
DAXPY	1014	2234	0	13
DOT	1178	2434	0	13
RP size	9200	18400	30	40
Max. %	12.8	13.2	0.0	32.5

a loopback FIFO PE placed on the Remote FPGA. The average round-trip cycle count is 126.3, while remote loopback takes 23 clock cycles in RTL simulation. From these results, the local-to-remote and remote-to-local transmission requires about 100 clock cycles, or 50 cycles (=512.0ns) for one-way trip. This is reasonably fast for a 1.5m cable and SerDes on both ends.

The figure also shows the latency jitter is no more than ±10 clock cycles, so the FPGA-to-FPGA interconnection doesn't require large buffers to manage the latency jitter.

4.4 Partial Reconfiguration

The STREAM/STREAM2 benchmark described above is implemented to partial reconfiguration to load its math kernels: the kernel modules are dynamically loaded on the FPGA during the benchmark execution. Figure 9 shows the FPGA floorplan, with a rectangular reconfiguration partition in X1Y4 clock region. Although the reconfiguration partition is about 1/20 of the total FPGA area, it's enough large to hold a math kernel pipeline. Of course, placing much larger partition is possible because current resource utilization is quite low.

The length of full configuration bitstream for XC7K325T FPGA is 11,443,612 bytes, and the partial bitstream for this reconfiguration partition is 1,362,740 bytes. The ICAP PE and ICAPE2 configuration access port accepts any configuration bitstream at 400MB/s (32bit at 100MHz), so the partial bitstream is loaded in about 3.4msec.

ICAP PE also works on the Remote FPGA through the FPGA-to-FPGA serial link, so the reconfiguration partitions of all FPGAs can be reprogrammed by host program.

4.5 Math kernel example: Diffusion Equation

The main aim this project is to build a general-purpose multi FPGA accelerator for stream-oriented scientific computing. Stream-oriented computing is an popular FPGA application, since DNA sequence matching algorithm is implemented on Splash 2[12] in early 90's. The approach is still

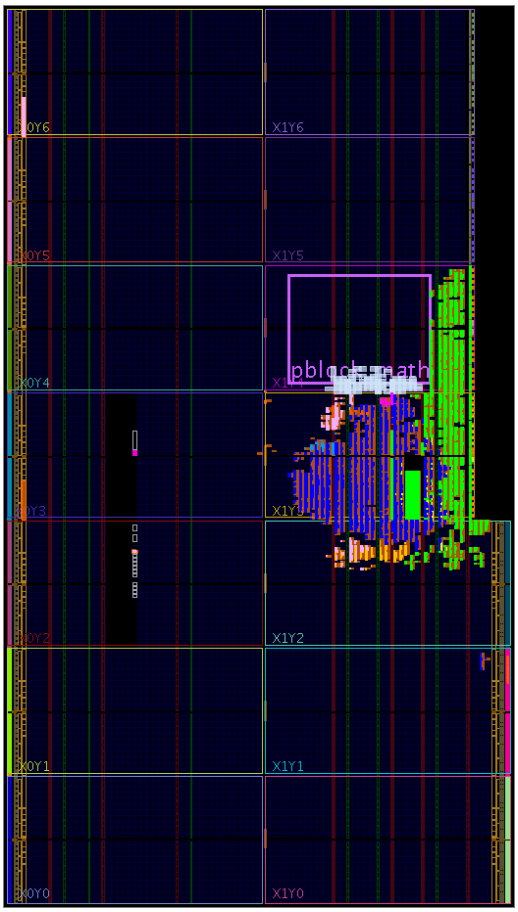


Figure 9: Local FPGA Floorplan

very powerful with modern large-scale FPGAs and floating-point arithmetic: for example, Sano's work[6] with Stratix V FPGA achieved much higher performance than GPUs in Tsunami simulation.

To verify the computational power this Host-FPGA infrastructure, a Stream PE to solve simplified 2D diffusion equation is implemented. On a regular, square grid and 4 neighbor cell values (x_u, x_d, x_l and x_r where up, down, left and right respectively), the center cell value x_c in next time step x_{c1} is calculated as:

$$x_{c1} = x_c + k((x_u + x_d + x_l + x_r) - 4x_c)$$

Because this equation contains 5 add/subtract operations and 2 multiplications, the SPE has also 5 adder/subtractors and 2 multipliers, all using fully-pipelined double precision

Table 3: Results from Diffusion Equation SPEs

SPEs	LUT	FF	BRAM	DSP	GFLOPS
1	5,888	9,096	18.5	35	1.75
2	11,290	17,551	18.5	70	3.50
4	22,084	34,462	18.5	140	6.99
8	41,377	68,284	18.5	280	13.95
16	86,957	135,940	18.5	560	27.98
24	130,200	203,846	18.5	840	41.95

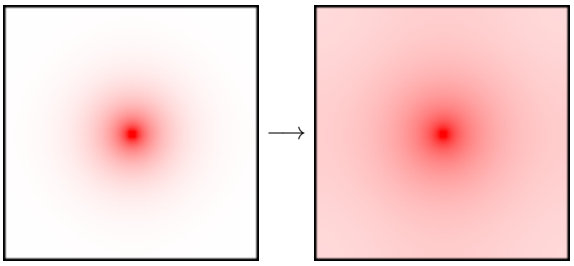


Figure 10: An example of results from diffusion equation

floating point cores. Figure 10 is an example of the result, diffusing from the center of the area.

The SPE has a shift register to extract x_u, x_d, x_l, x_r and x_c continuously, so the SPE can solve 100×100 grid in fully pipelined manner. As an SPE calculates cell values in next time step, n-sequentially concatenated SPEs calculate cell values after n time step. This enables more throughput without memory access or host communication overhead.

Table 3 summarizes the SPEs' resource utilization and performance in GFLOPS. While he DSP usage is 100% with 24 SPEs, there are still more LUTs and BRAMs. Usage of BRAM is constant in all implementations because BRAM is placed in only at input and output buffers. GFLOPS values on the table is best achieved ones, frequent intermediate result transfers to the host (usually once less than 10^4 time steps) suffers the throughput about 10 to 20%.

The SPE is completely pipelined, so the bandwidth between any two adjacent SPEs is always constant. This enables to extend its implementation to multiple FPGAs with linear performance gain. Although these SPEs are implemented on Local FPGA, it will be extended to use both Local and Remote FPGAs in near future.

5. CONCLUSION AND FUTURE WORK

In this paper, the CPU-FPGA hybrid cluster architecture for stream oriented scientific computation was proposed and evaluated with the prototype system. To enable migration to other FPGAs or boards, the device dependent part is carefully separated in the RTL design. Also, it's built using FIFO-based PCIe and high-speed serial interface IP cores, that are available for most middle-range and high-end FPGAs. As the result, most RTL code is common for 3 FPGAs in the prototype system, except the IP cores and their wrapper modules.

Both Host-to-FPGA and FPGA-to-FPGA link performance is evaluated with STREAM/STREAM2 benchmark, including partial reconfiguration of the FPGAs. The stream computation performance was also evaluated with simple 2D diffusion equation.

The results show insufficient bandwidth on PCIe when the data transfer size is small. To avoid this problem, we're planning to test RIFFA[13] or that kind of more powerful PCI Express interface. Also, the cluster will be expanded with multiple Kintex Ultrascale FPGA boards soon. This will also draw the serial line rate up to 10Gbps. With the system expansion, we'll develop methods to active multiple SPEs simultaneously and to enable SPEs spanning multiple FPGAs. We'll also investigate high-level synthesis method for Stream PE design.

Acknowledgments

This work is supported by JSPS KAKENHI Grant Number JP16K00078.

This work is supported by VLSI Design and Education Center(VDEC), the University of Tokyo in collaboration with Cadence Design Systems, Inc. The authors also thank to Xilinx University Program.

6. REFERENCES

[1] A. Putnam *et.al*: "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services.", in Proceedings of the 2014 International Symposium on Computer Architecture (ISCA), Jun.2014.

[2] SuperVessel Cloud: `https://ptopenlab.com/`

[3] Amazon EC2 F1 Instances: `https://aws.amazon.com/ec2/instance-types/f1/`

[4] O. Mencer, *et.al*: "Cube[1: A 512-FPGA cluster.", in Proceedings of the 5th Southern Conference on Programmable Logic (SPL), May.2009.

[5] A. George, *et.al*: "Novo-G: at the Forefront of Scalable Reconfigurable Supercomputing." Computing in Science and Engineering, Vol.13, No.1, pp.82–86. 2011.

[6] K. Sano *et.al*: "Stream Computation of Shallow Water Equation Solver for FPGA-based 1D Tsunami Simulation.", in Proceedings of the 2015 International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART), Jun.2016.

[7] K. Fleming, *et.al*: "The LEAP FPGA Operating System." in Proceedings of the 2014 International Conference on Field-Programmable Logic and Applications (FPL), Sep. 2014.

[8] Xillybus Ltd.: "Xillybus IP core product brief." `http://xillybus.com/downloads/xillybus_product_brief.pdf`, v.1.8, Jan.2016.

[9] Xilinx Inc.: "Aurora 64B/66B v11.1 Product Guide." Oct.2016.

[10] J. McCalpin: "Memory Bandwidth and Machine Balance in Current High Performance Computers", IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, Dec.1995.

[11] J. McCalpin: "The STREAM2 Home Page." `http://www.cs.virginia.edu/stream/stream2/` .

[12] J. M. Arnold, *et.al*: "Splash 2.", In Proceedings of 4th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 316–322, 1992.

[13] M. Jacobsen *et.al*: "RIFFA 2.1: A reusable integration framework for FPGA accelerators." ACM Transactions on Reconfigurable Technology and Systems (TRETS), Sep.2015.