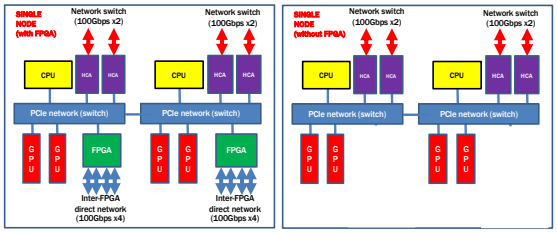
Cygnus: A Multi-Hybrid Supercomputing Platform with GPUs and FPGAs

Taisuke Boku[1], Ryohei Kobayashi[1], Norihisa Fujita[1], Hideharu Amano[2], Kentaro Sano[3], Toshihiro Hanawa[4], Yoshiki Yamaguchi[1]

1: University of Tsukuba, Japan
2: Keio University, Japan
3: RIKEN Center for Computational Science, Japan
4: The University of Tokyo, Japan

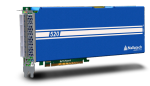
Multi-Hybrid Accelerated Computing

- Combining goodness of different type of accelerators: GPU + FPGA
  - GPU is still an essential accelerator for simple and large degree of parallelism to provide ~10 TFLOPS peak performance
  - FPGA is a new type of accelerator for application-specific hardware with programmability and speeded up based on pipelining of calculation
  - FPGA is good for external communication between them with advanced high speed interconnection up to 100Gbps x4chan.
- Next supercomputer "Cygnus" will be deployed
  - Test operation starts in April 2019, public operation starts in May 2019
  - 2x Intel Xeon CPUs, 4x NVIDIA V100 GPUs, 2x Intel Stratix10 FPGAs
  - Deneb: 46 CPU+GPU nodes
  - Albireo: 32 CPU+GPU+FPGA nodes with 2D-torus dedicated network for FPGAs (100Gbpsx4)



Albireo node (x32) Deneb node (x46)

Target GPU: NVIDIA Tesla V100

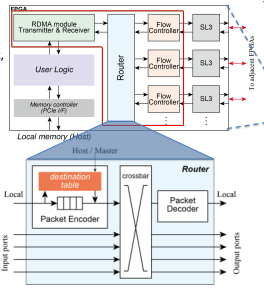


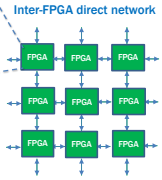
Target FPGA: Nallatech 520N

Specification of Cygnus

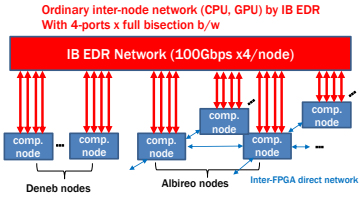
Item	Specification
Peak performance	2.4 PFLOPS DP (GPU: 2.2 PFLOPS, CPU: 0.2 PFLOPS, FPGA: 0.6 PFLOPS SP) ⇨ enhanced by mixed precision and variable precision on FPGA
# of nodes	78 (32 Albireo (GPU+FPGA) nodes, 46 Deneb (GPU-only) nodes) ⇨ 2 additional nodes will come, in total 80
Memory	192 GiB DDR4-2666/node = 256GB/s, 32GiB x 4 for GPU/node = 3.6TB/s
CPU / node	Intel Xeon Gold (SKL) x2 sockets
GPU / node	NVIDIA V100 x4 (PCIe)
FPGA / node	Intel Stratix10 x2 (each with 100Gbps x4 links/FPGA and x8 links/node)
Global File System	Lustre, RAID6, 2.5 PB
Interconnection Network	Mellanox InfiniBand HDR100 x4 (two cables of HDR200 / node) 4 TB/s aggregated bandwidthj
Programming Language	CPU: C, C++, Fortran, OpenMP, GPU: OpenACC, CUDA FPGA: OpenCL, Verilog HDL
System Vendor	NEC

- FPGA design plan
  - Router
    - For the dedicated network, this impl. is mandatory.
    - Forwarding packets to destinations
  - User Logic
    - OpenCL kernel runs here.
    - Inter-FPGA comm. can be controlled from OpenCL kernel.
  - SL3
    - SerialLite III : Intel FPGA IP
    - Including transceiver modules for Inter-FPGA data transfer.
    - Users don't need to care





64 FPGAs on Albireo nodes are connected directly as 2D-Torus configuration without Ethernet sw.

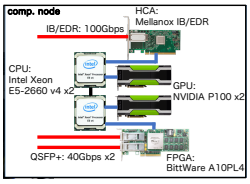


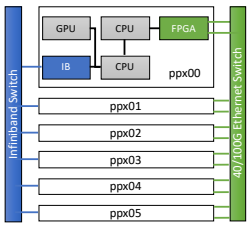
Ordinary inter-node communication channel for CPU and GPU, but they can also request it to FPGA

Accelerator in Switch (AiS)

- Accelerator in Switch (AiS) is a concept proposed by Prof. Amano, Keio University, Japan
  - It couples communication and computations tightly
  - FPGAs can act as both of computation accelerators and network switches
- FPGA programming cost using Hardware Description Language (HDL) is very expensive
- Due to improvement of High Level Synthesis (HLS), programming cost of FPGA is decreasing
  - No HDL code is required
  - Application programmers can program FPGAs
- We consider we can realize AiS system using FPGAs

- Pre-PACS-X (PPX) is a test-bed system in Center for Computational Sciences, University of Tsukuba
  - It is a prototype of the next generation system of their PACS series supercomputer
  - Each node has 2 CPUs, 2 GPUs and 2 FPGAs
  - Not only InfiniBand network for CPUs but also 40GbE network for FPGAs



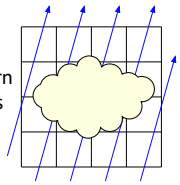


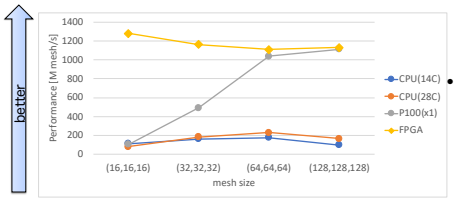
ART on FPGA

- Accelerated Radiative transfer on grids Oct-Tree (ARGOT) has been developer in Center for Computational Sciences, University of Tsukuba
  - Authentic Radiative Transfer (ART) method is one of algorithms used in ARGOT and dominant part (90% or more of computation time) of ARGOT program
- ART is ray tracing based algorithm
  - problem space is divided into meshes and reactions are computed on each mesh
  - ART method computes radiative intensity on each mesh as shows as formula (1)

$$I_\nu^{out}(\hat{n}) = I_\nu^{in}(\hat{n})e^{-\Delta\tau_\nu} + S_\nu(1 - e^{-\Delta\tau_\nu}) \quad (1)$$

- Memory access pattern for mesh data is varies depending on ray's direction
  - Memory access pattern for mesh data is varies depending on ray's direction
  - Not suitable for SIMD architecture





Inter-FPGA communication

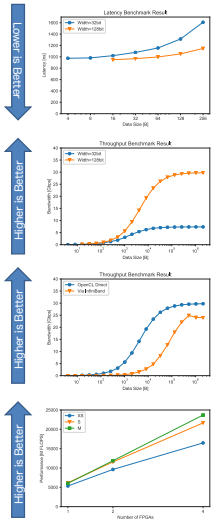
- Channel over Ethernet (CoE)
  - CoE enables OpenCL code communicate with other FPGAs on different nodes
  - Extending Intel's channel mechanism to external communications
  - Pipeline manner: sending/receiving data from/to compute pipeline directly

sender code on FPGA1 — CoE Backends — receiver code on FPGA2

```
sender(__global float* restrict x, int n) {
  for (int i = 0; i < n; i++) {
    float v = x[i];
    write_channel_intel(simple_out, v);
  }
}
```

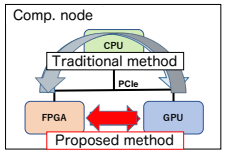
```
receiver(__global float* restrict x, int n) {
  for (int i = 0; i < n; i++) {
    float v = read_channel_intel(simple_in);
    x[i] = v;
  }
}
```

- CoE supports 32bit or 128bit width channels
  - On every cycle, we can send or receive a value through a CoE channel
- Pingpong benchmark results:
  - 950ns min. latency
  - 29.77Gbps max. throughput
- Much faster than the traditional method
  - OpenCL direct comm. vs. through CPU via IB (store-and-forward)
- Himeno Benchmark (3D CFD minibench)
  - 19-point 3D stencil
  - x3.93 speedup compared to 1 FPGA



OpenCL-enabled GPU-FPGA DMA

- The FPGA (OpenCL kernel) autonomously performs the DMA-based data movement (not through CPU)
- I/O Channel API is used to control the functionality



Example code to invoke the DMA

OpenCL Kernel code

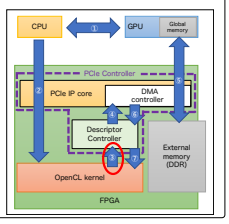
```
#pragma OPENCL EXTENSION cl_intel_channels : enable
channel cldesc_t fpga_dma __attribute__((depth(0)))
                          __attribute__((io("chan_fpga_dma")));

typedef struct __attribute__((packed)) cldesc {
  ulong src;
  ulong dst;
  uint  id_and_len;
} cldesc_t;
...
cldesc_t desc;
...
desc.src      = PADDR;
desc.dst      = (ulong)&src_array[0];
desc.id_and_len = id_and_len;
write_channel_intel(fpga_dma, desc);
```

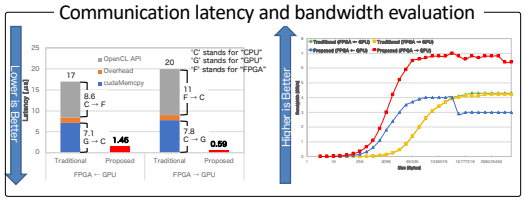
Descriptor definition

Setting data

Sending a descriptor to the Descriptor Controller



Communication latency and bandwidth evaluation



Future Work

- How FPGA knows GPU computation completion?
  - A sophisticated synchronization mechanism is needed
- No one wants to do multilingual programming!! (CUDA, OpenCL, etc.)
  - needs a comprehensive programming framework enabling the programming in a single language (w/ OpenACC)
- Combining inter-FPGA comm. and GPU-FPGA DMA

ACKNOWLEDGEMENT This work was supported in part by MEXT as "Next Generation High-Performance Computing Infrastructures and Applications R&D Program" (Development of Computing- Communication Unified Supercomputer in Next Generation). This research was also supported (in part) by Multidisciplinary Cooperative Research Program in CCS, University of Tsukuba, and JSPS KAKENHI Grant Number 18H03246. We also thank Intel University Program for providing us both of hardware and software.