# Cygnus: GPU meets FPGA for HPC

Taisuke Boku

Director, Center for Computational Sciences

University of Tsukuba

in collaboration with R. Kobayashi, N. Fujita, Y. Yamaguchi and M. Umemura
at CCS, U. Tsukuba

# Agenda

- Introduction
- Today's accelerated supercomputing
- Multi-Hybrid accelerated supercomputing
- Cygnus supercomputer in U. Tsukuba
- How to use and program Cygnus
- Summary

# History of PAX (PACS) series at U. Tsukuba

- 1977: research started by T. Hoshino and T. Kawai
- 1978: PACS-9 (with 9 nodes) completed
- 1996: CP-PACS, the first vendor-made supercomputer at CCS, ranked as #1 in TOP500

1978
1st gen: PACS-9

1980
2nd gen. PACS-32

1989
5th gen, QCDPAX

1996
6th gen: CP-PACS
Ranked #1 in TOP500

2006
7th gen: PACS-CS

2012~2013
8th gen: GPU cluster HA-PACS

2014
9th gen: COMA

| Year | Name | Performance |
|---|---|---|
| 1978 | PACS-9 | 7 KFLOPS |
| 1980 | PACS-32 | 500 KFLOPS |
| 1983 | PAX-128 | 4 MFLOPS |
| 1984 | PAX-32J | 3 MFLOPS |
| 1989 | QCDPAX | 14 GFLOPS |
| 1996 | CP-PACS | 614 GFLOPS |
| 2006 | PACS-CS | 14.3 TFLOPS |
| 2012~13 | HA-PACS | 1.166 PFLOPS |
| 2014 | COMA (PACS-IX) | 1.001 PFLOPS |
| 2019 | Cygnus (PACS-X) | 2.5 PFLOPS |

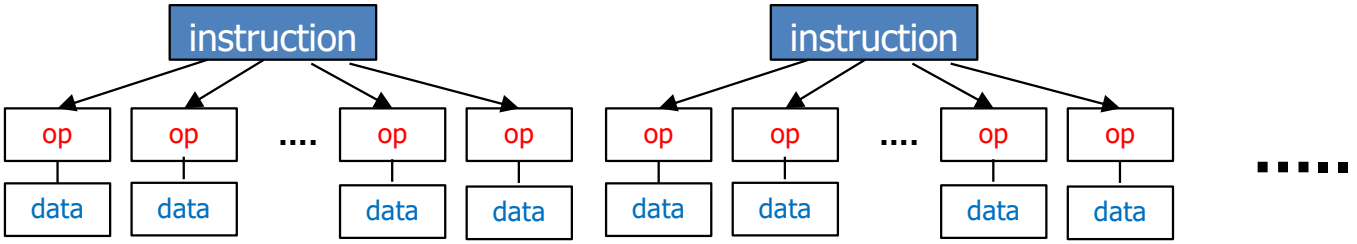
- *co-design* by computer scientists and computational scientists toward "practically high speed computer"
- Application-driven development
- Sustainable development experience

# Accelerators in HPC

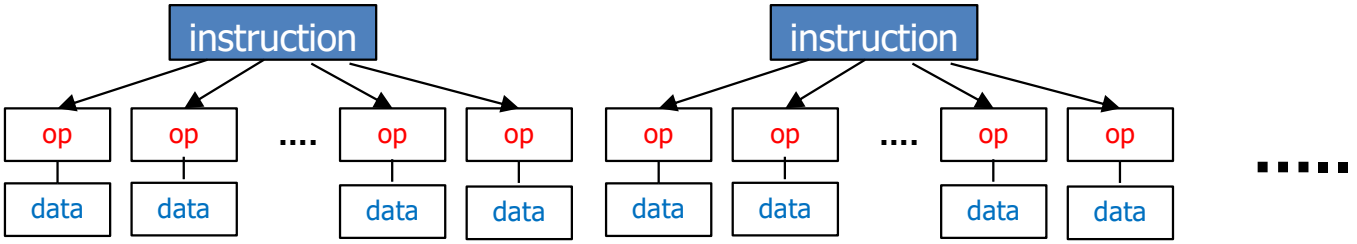
- Traditionally...
  - Cell Broadband Engine, ClearSpeed, GRAPE. ...
  - then GPU (most popular)
- Is GPU perfect ?
  - good for many applications (replacing vector machines)
  - depending on very wide and regular parallelism
    - large scale SIMD (STMD) mechanism in a chip
    - high bandwidth memory (HBM, HBM2) and local memory
  - insufficient for cases with...
    - not enough parallelism
    - not regular computation (warp splitting)
    - frequent inter-node communication (kernel switch, go back to CPU)

NVIDIA Tesla V100 (Volta)
with PCIe interafce

# GPU performance: obtained by large scale SIMD type parallelism

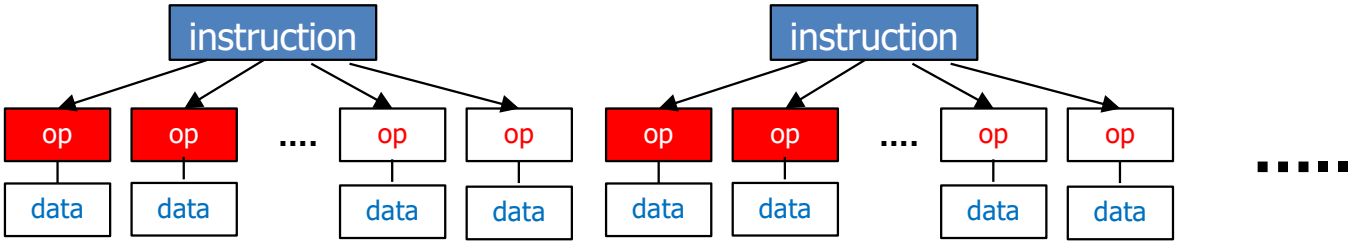


# GPU performance: branch condition degrades performance



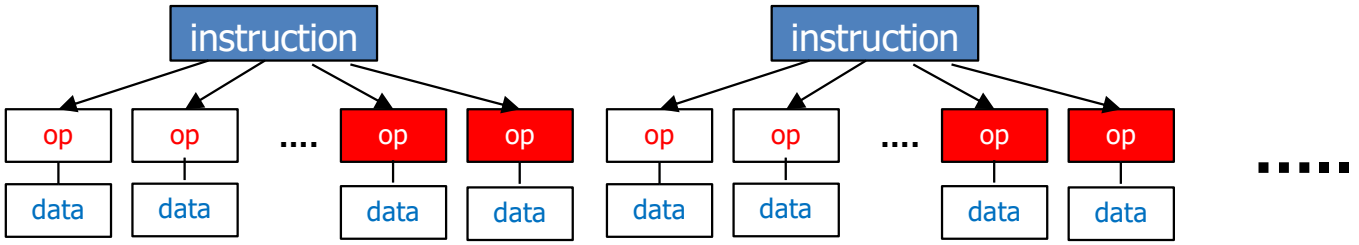

```
if (a[i] < x)
    b[i] = a[i];
else
    b[i] = 0.0;
```

# GPU performance: branch condition degrades performance



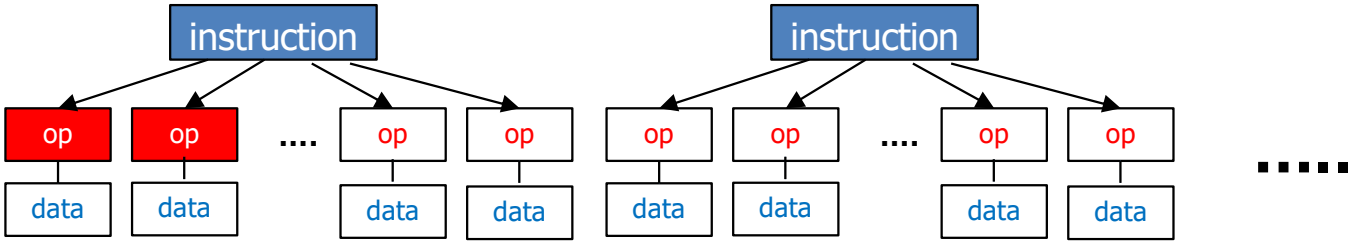

```
if (a[i] < x)
    b[i] = a[i];
else
    b[i] = 0.0;
```

# GPU performance: branch condition degrades performance



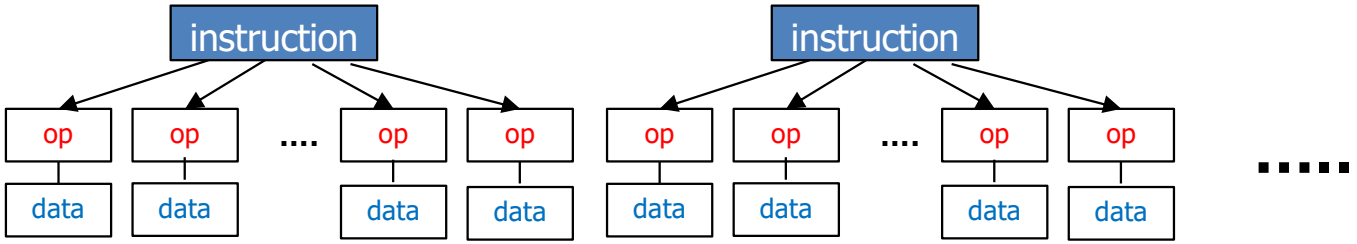

```
if (a[i] < x)
    b[i] = a[i];
else
    b[i] = 0.0;
```

# GPU performance: low parallelism




```
for( i = 0; i < 2; i++)
....
```

# GPU performance: frequent communication




```
(in host code)
MPI_xxx();
GPU_kernel1();
MPI_yyy();
GPU_kernel2();
....
```

# FPGA (Field Programmable Gate Array)

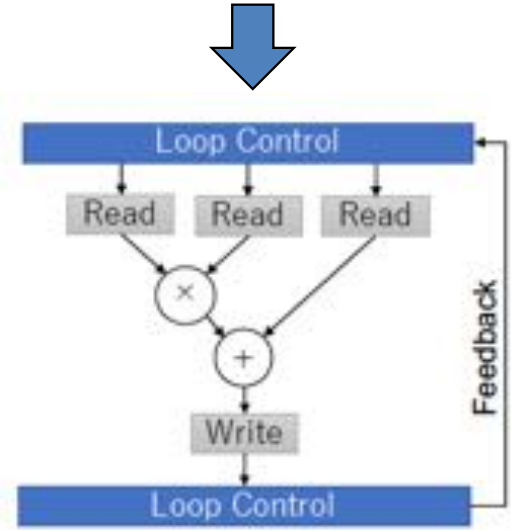
- Goodness of recent FPGA for HPC
  - True codesigning with applications (essential)
  - Programmability improvement: OpenCL, other high level languages
  - High performance interconnect: 100Gb
  - Precision control is possible
  - Relatively low power
- Problems
  - Programmability: OpenCL is not enough, not efficient
  - Low standard FLOPS: still cannot catch up to GPU
    -> “never try what GPU works well on”
  - Memory bandwidth: 1-gen older than high end CPU/GPU
    -> be improved by HBM (Stratix10)

Nallatech 520N with Intel Stratix10 FPGA equipped with 4x 100Gbps optical interconnection interfaces

# What is FPGA ?

- FPGA – Field Programmable Gate Array
- Reconfigurable logic circuit based on user description (low or high level) where all described logic elements are implemented to the circuit with network, gate and flip-flop
- Low level language has been used so far, recently HLS (High Level Synthetic) is available such as C, C++, OpenCL

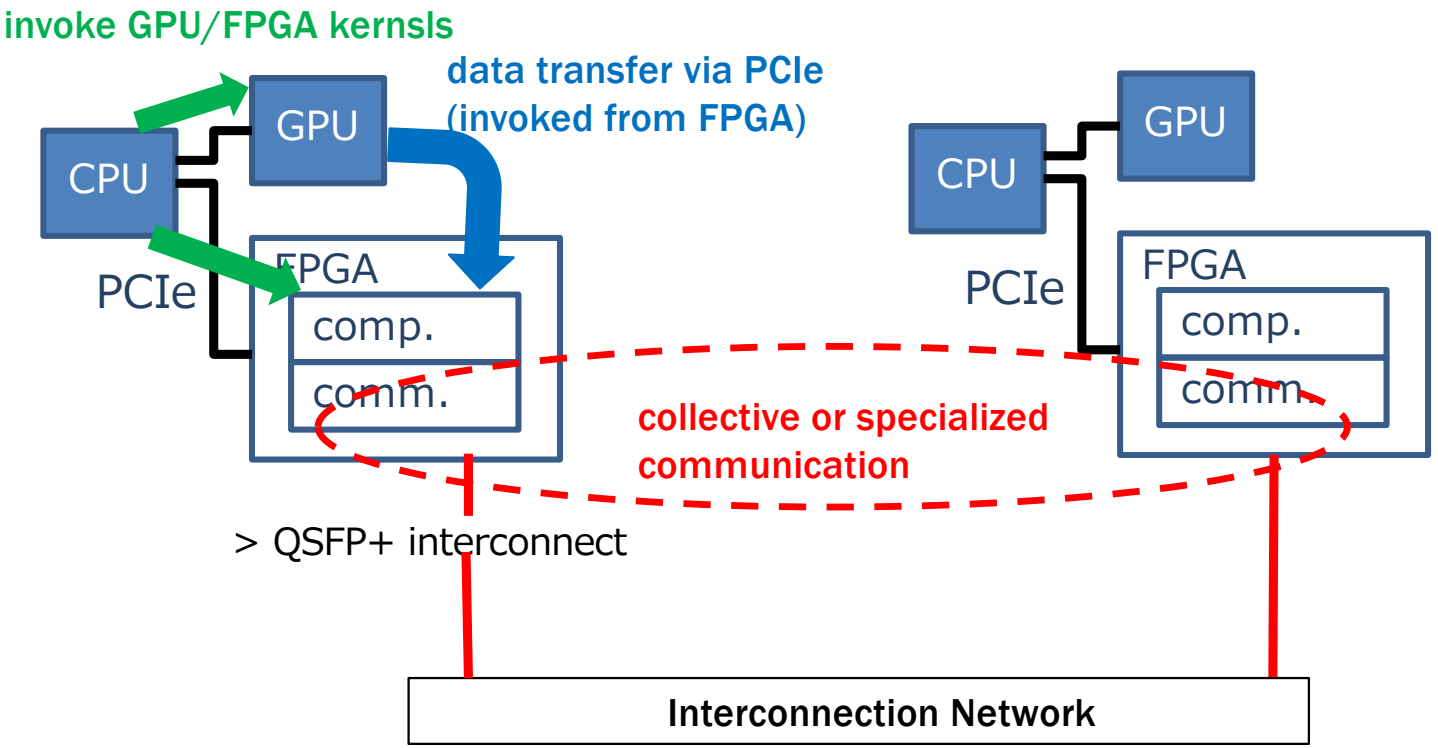
```
for (int i = 0; i < n; i++) {
    d[i] = a[i] * b[i] + c[i]
}
```




- all “n” elements of computation is pipelined
- circuit frequency is based on the complexity and length of each element path

# AiS: conceptual model of Accelerator in Switch

- FPGA can work both for computation and communication in unified manner
- GPU/CPU can request application-specific communication to FPGA

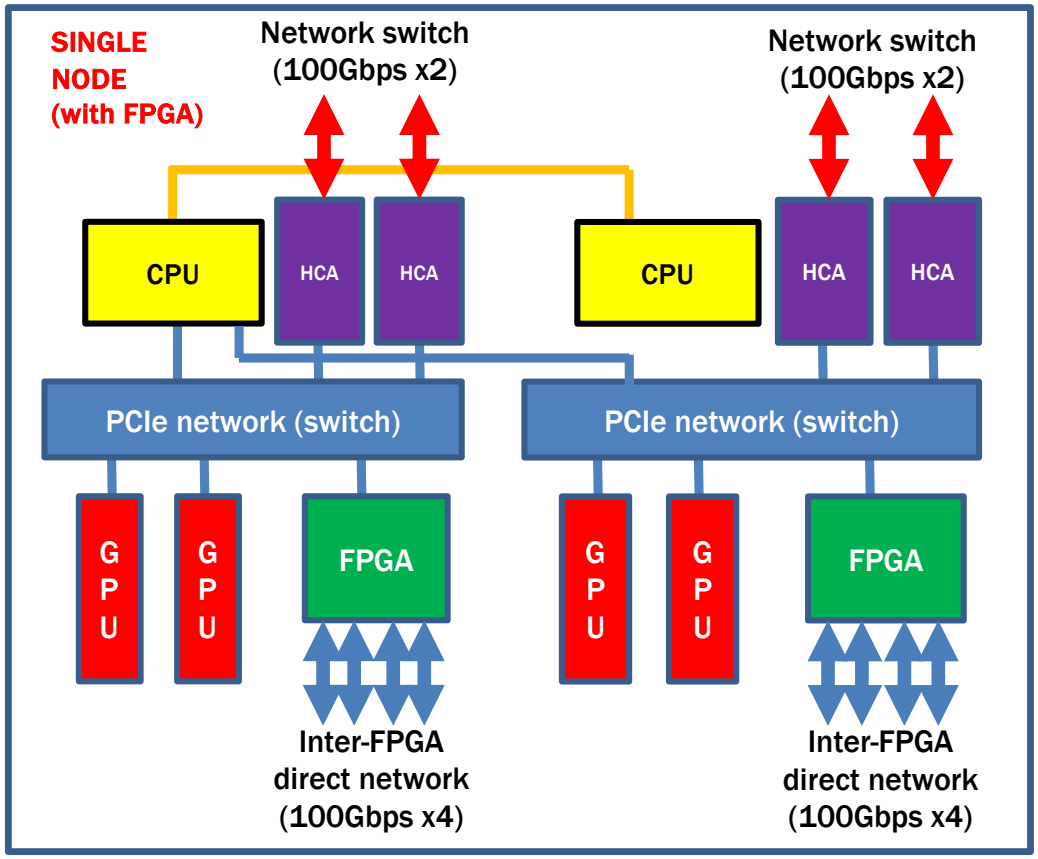


# Cygnus Overlook (CCS, U. Tsukuba, April 2019)



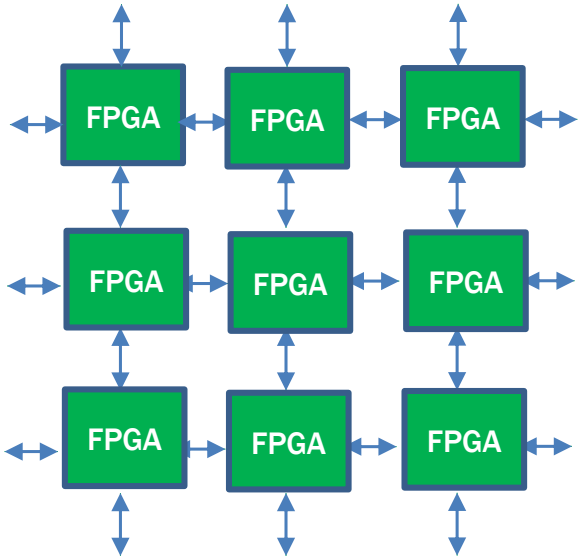
# Single node configuration (Albireo) of Cygnus cluster

- Each node is equipped with both IB EDR and FPGA-direct network
- Some nodes are equipped with both FPGAs and GPUs, and other nodes are with GPUs only



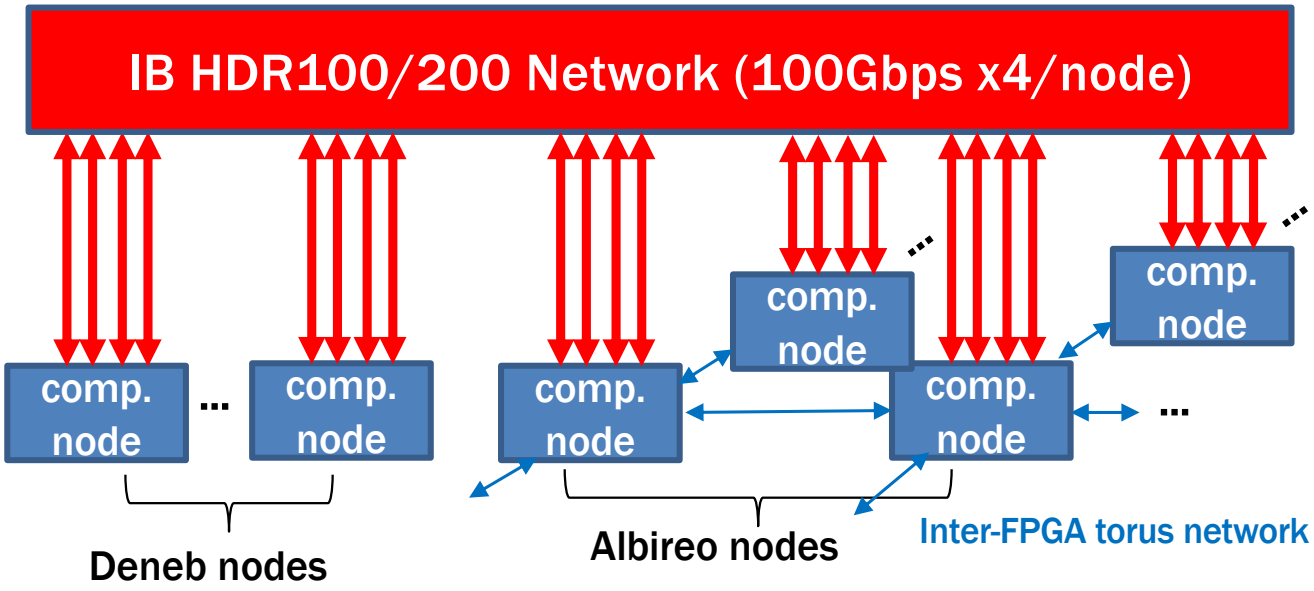
# Two types of interconnection network

Inter-FPGA direct network
(only for Albireo nodes)

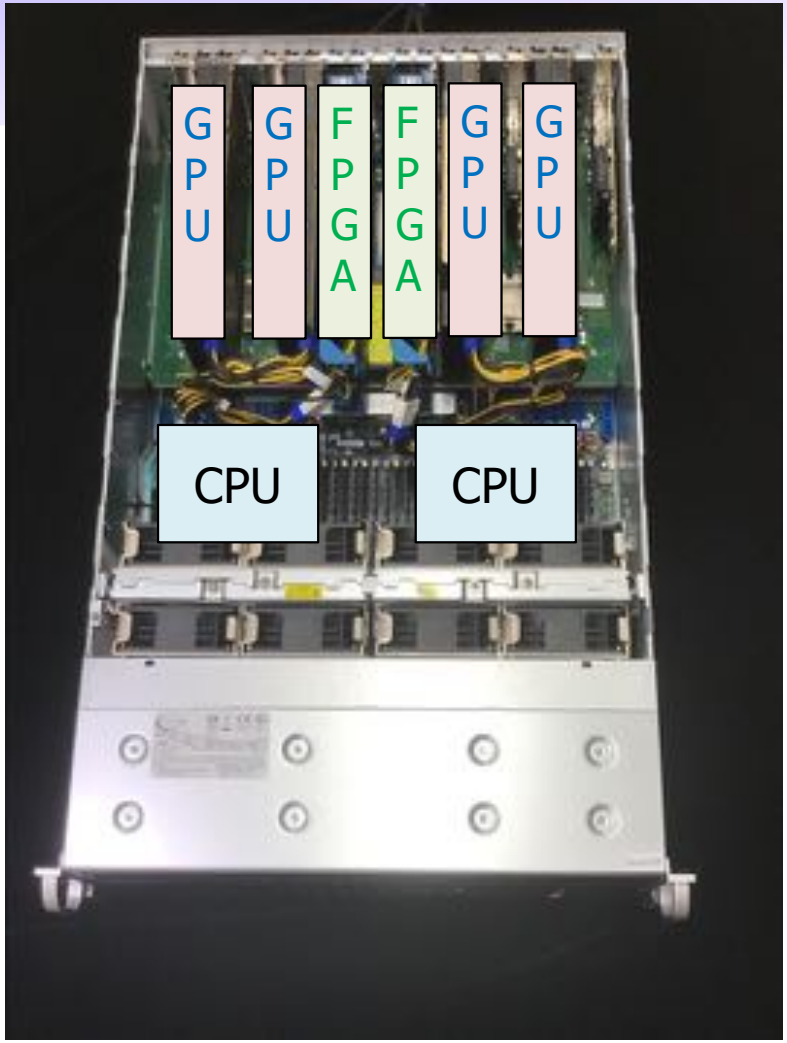


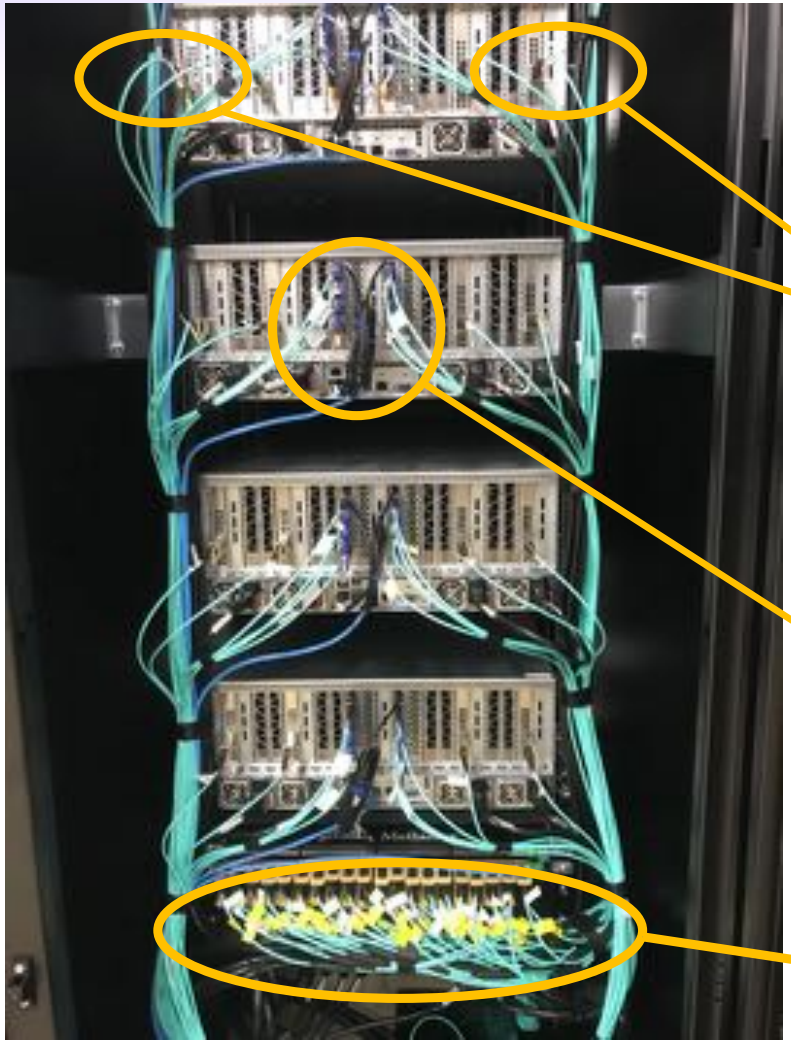

64 of FPGAs on Albireo nodes (2 FPGAS/node) are connected by 8x8 2D torus network without switch

InfiniBand HDR100/200 network for parallel processing communication and shared file system access from all nodes




For all computation nodes (Albireo and Deneb) are connected by full-bisection Fat Tree network with 4 channels of InfiniBand HDR100 (combined to HDR200 switch) for parallel processing communication such as MPI, and also used to access to Lustre shared file system.



IB HDR100 x4
⇨ HDR200 x2

100Gbps x4
FPGA optical
network

IB HDR200
switch (for
full-bisection
Fat-Tree)

# Specification of Cygnus

| Item | Specification |
|---|---|
| Peak performance | 2.4 PFLOPS DP<br>(GPU: 2.24 PFLOPS, CPU: 0.16 PFLOPS + FPGA: 0.64 SP FLOPS) |
| # of nodes | 80 (32 Albireo nodes, 48 Deneb nodes) => 320x V100 + 64x Stratix10 |
| CPU / node | Intel Xeon Gold x2 sockets |
| GPU / node | NVIDIA Tesla V100 x4 (PCIe) |
| FPGA / node | Nallatech 520N with Intel Stratix10 x2 (each with 100Gbps x4 links) |
| NVMe | Intel NVMe 1.6TB, driven by NVMe-oF Target Offload |
| Global File System | DDN Lustre, RAID6, 2.5 PB |
| Interconnection Network | Mellanox InfiniBand HDR100 x4 = 400Gbps/node (SW=HDR200) |
| Total Network B/W | 4 TB/s |
| Programming Language | CPU: C, C++, Fortran, OpenMP<br>GPU: OpenACC, CUDA FPGA: OpenCL, Verilog HDL |
| System Integrator | NEC |

# How to open such a complicated system to application users ?

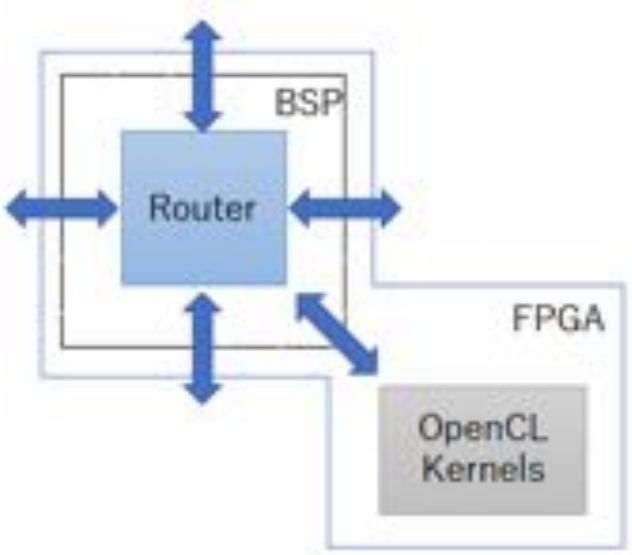
- OpenCL environment is available
  - ex) Intel FPGA SDK for OpenCL
  - basic computation can be written in OpenCL without Verilog HDL
- Current FPGA board is not ready for OpenCL on interconnect access
  - BSP (Board Supporting Package) is not complete for interconnect
    → we developed for OpenCL access
  - GPU/FPGA communication is very slow via CPU memory
- Our goal
  - enabling OpenCL description by users including inter-FPGA communication
  - providing basic set of HPC applications such as collective communication, basic linear library
  - providing 40G~100G Ethernet access with external switches for large scale systems

# CIRCUS

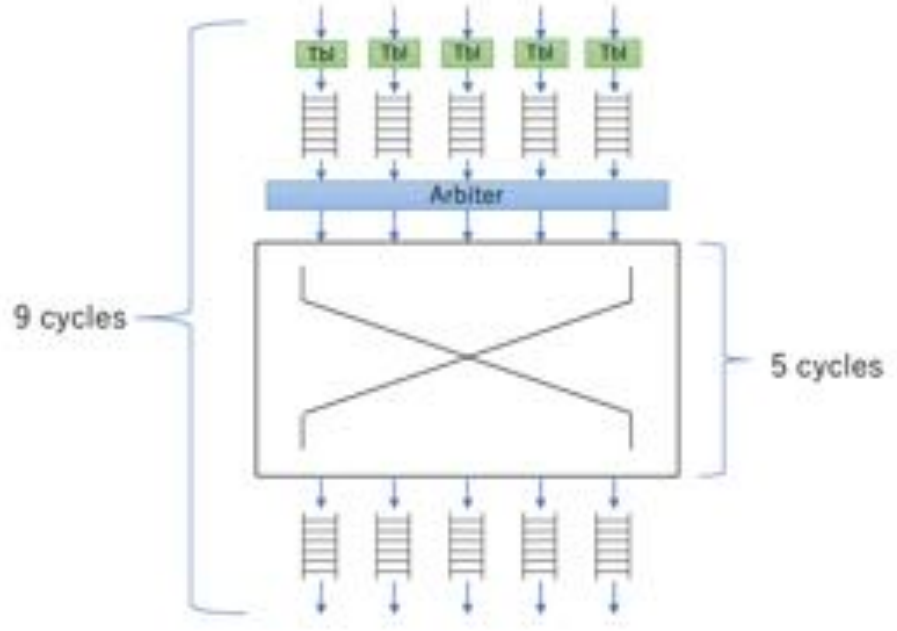
- FPGA is possible to combine computation and communication in a single framework of pipelined data stream
  - loop computation is pipelined according to the index
  - all the computation part is implemented on logic elements except buffering on memory
  - possible to access IP by chip provides (ex. Intel) for optical link driving
- making all to be programmable on OpenCL
  - scientific users never write Verilog HDL -> perhaps OK with OpenCL
  - key issue for practical HPC cluster: OpenCL-enabled features such ash
    - FPGA communication link
    - GPU/FPGA DMA

***CIRCUS***: Communication Integrated Reconfigurable CompUting System

# Optical link router in CIRCUS




Router written in Verilog HDL and implemented in BSP




packet queue and switching in CIRCUS router

# How to use

- simple data send/receive feature on OpenCL code
- OpenCL kernel and CIRCUS router is connected by Intel io-channel

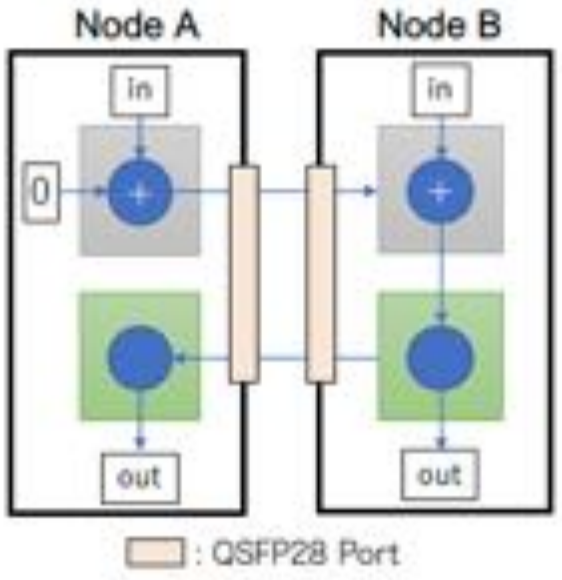
```
__kernel void sender(__global float* restrict x, int n){
  for(int i=0; i<n; i++){
    float v=x[i];
    write_channel_intel(simple_out, v);
  }
}
```

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

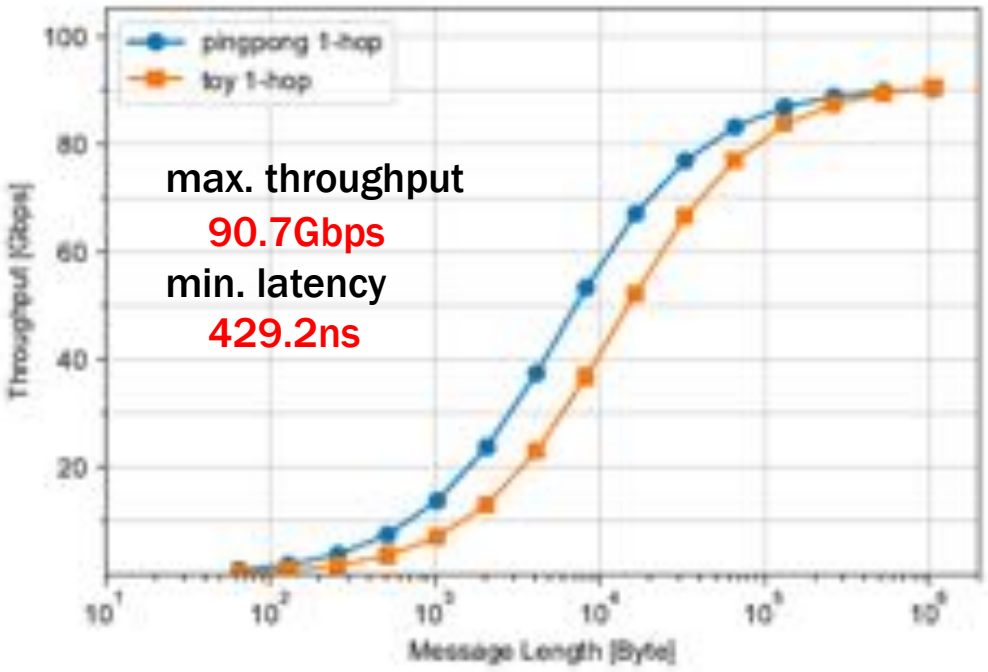
sender/receiver code on OpenCL to call CIRCUS communication

# Computation/communication pipelining on CIRCUS

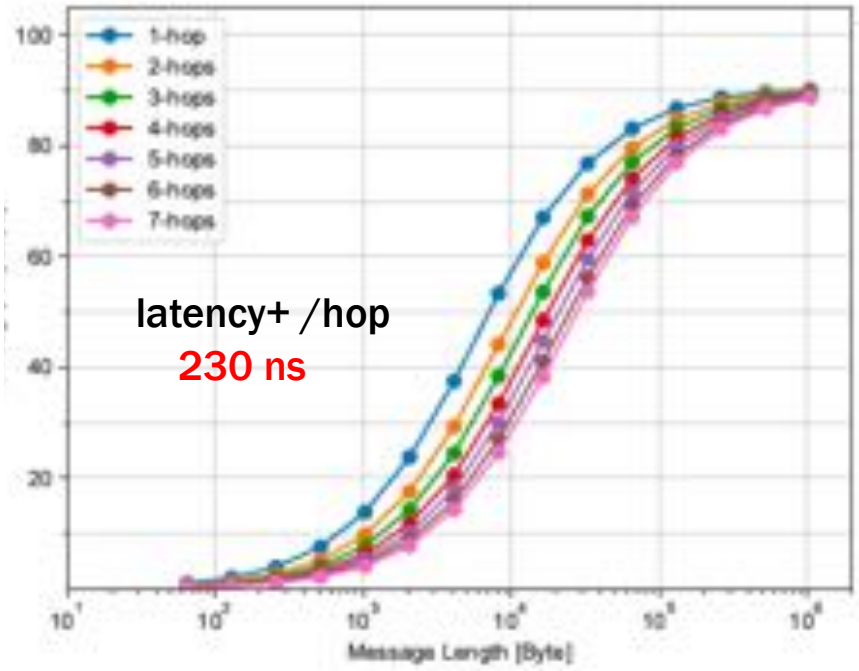
- we can pipeline computation and communication
- example: reduction on OpenCL
  - in “transmission kernel”, data is received then added to local data, and finally sent out to the neighboring FPGA
  - loop constructed code enables vector reduction
  - all in single pipeline, so throughput is not degraded but latency is increased



# Performance on Stratix10 in Cygnus

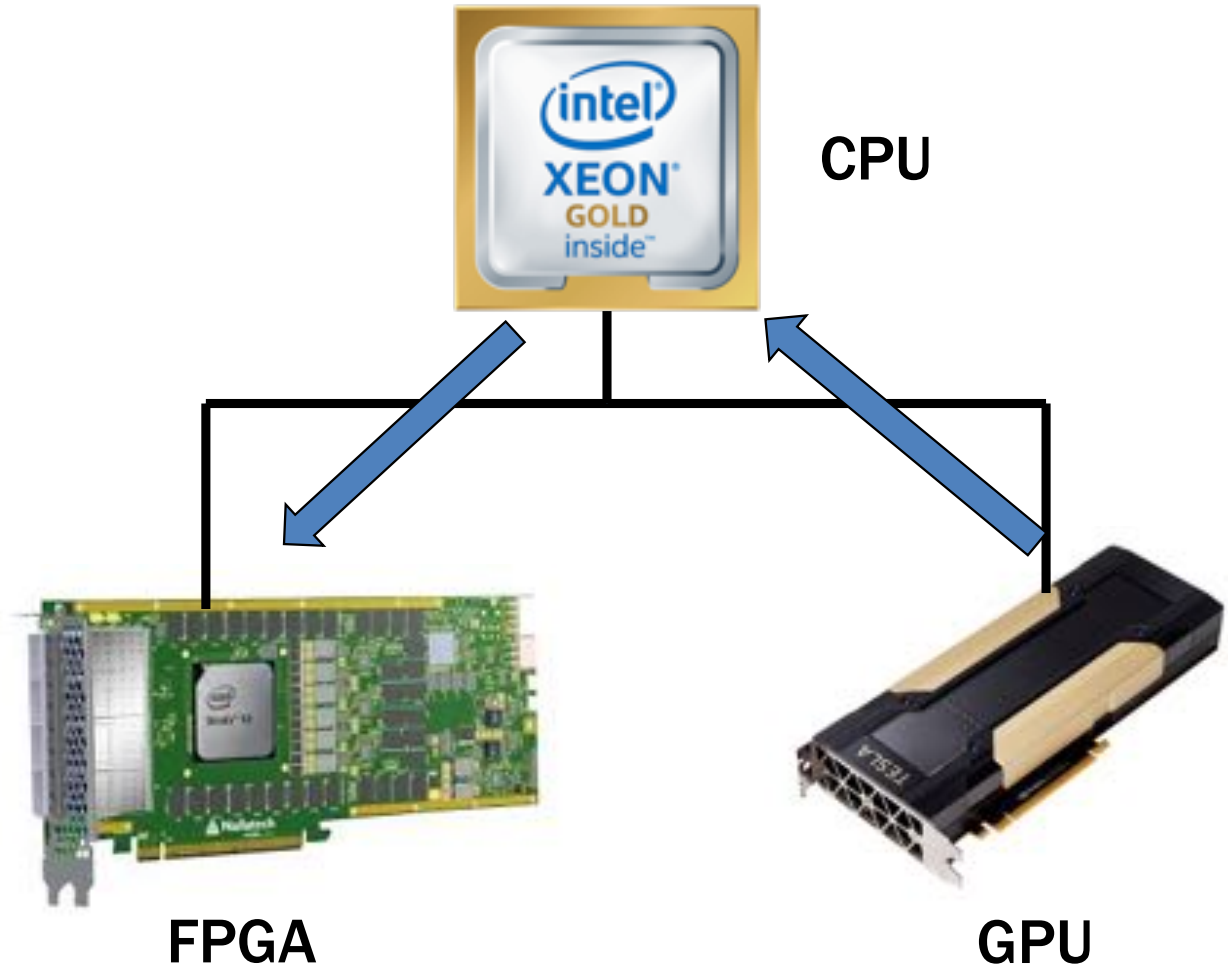



single ping-pong vs allreduce by OpenCL

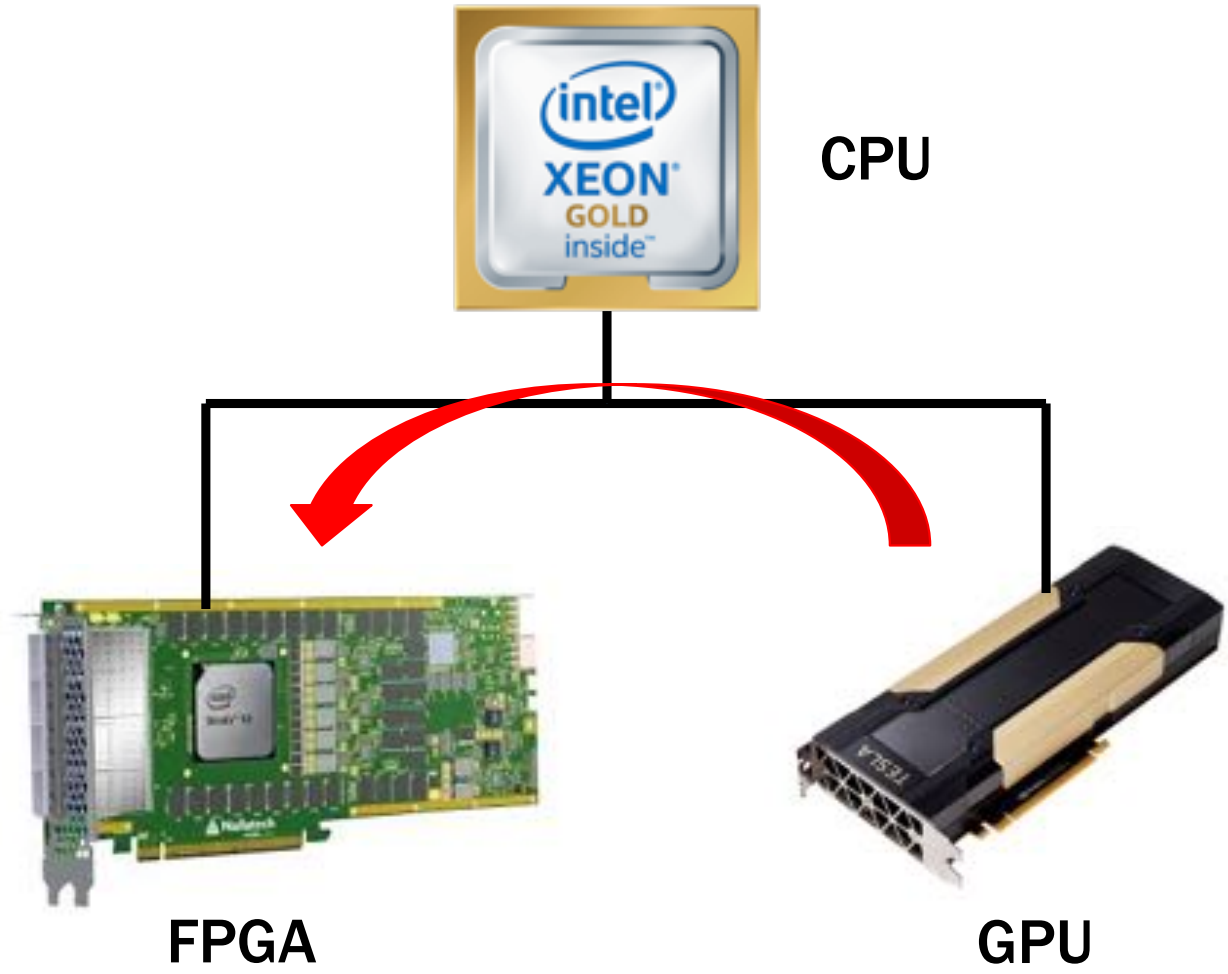


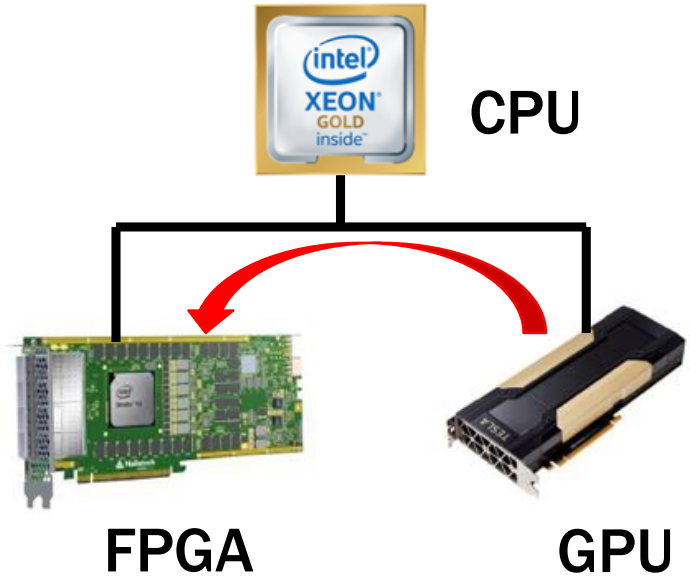

multi-hop ping-pong

# GPU-FPGA communication (via CPU memory)



# GPU-FPGA communication (DMA)

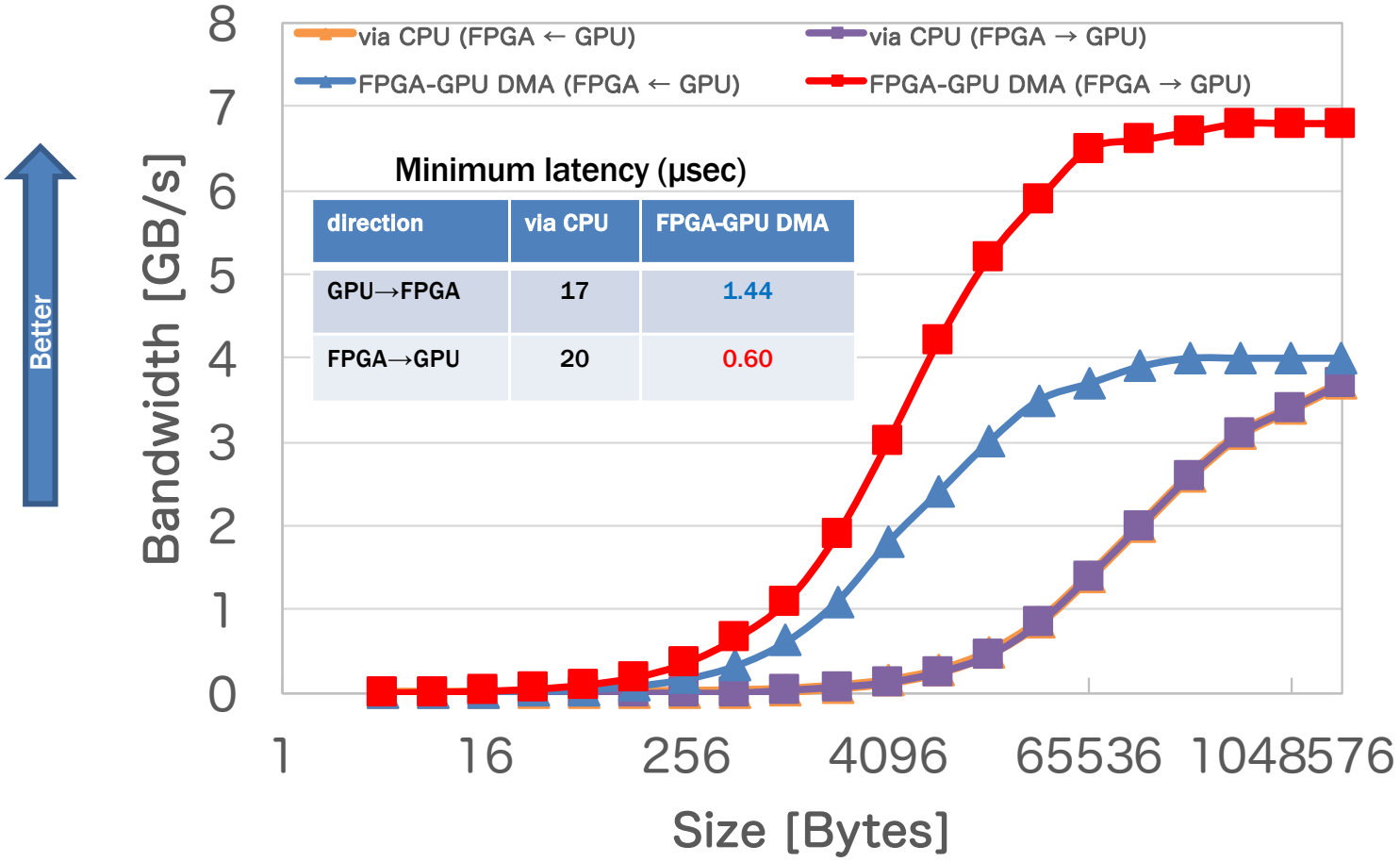





```
__kernel void fpga_dma(__global float *restrict fpga_mem,
                       const ulong gpu_memadr,
                       const uint id_and_len)
{
  cldesc_t desc;
  // DMA transfer GPU -> FPGA
  desc.src = gpu_memadr;
  desc.dst = (ulong)(&fpga_mem[0]);
  desc.id_and_len = id_and_len;
  write_channel_intel(fpga_dma, desc);
  ulong status = read_channel_intel(dma_stat);
}
```

GPU-to-FPGA DMA kick

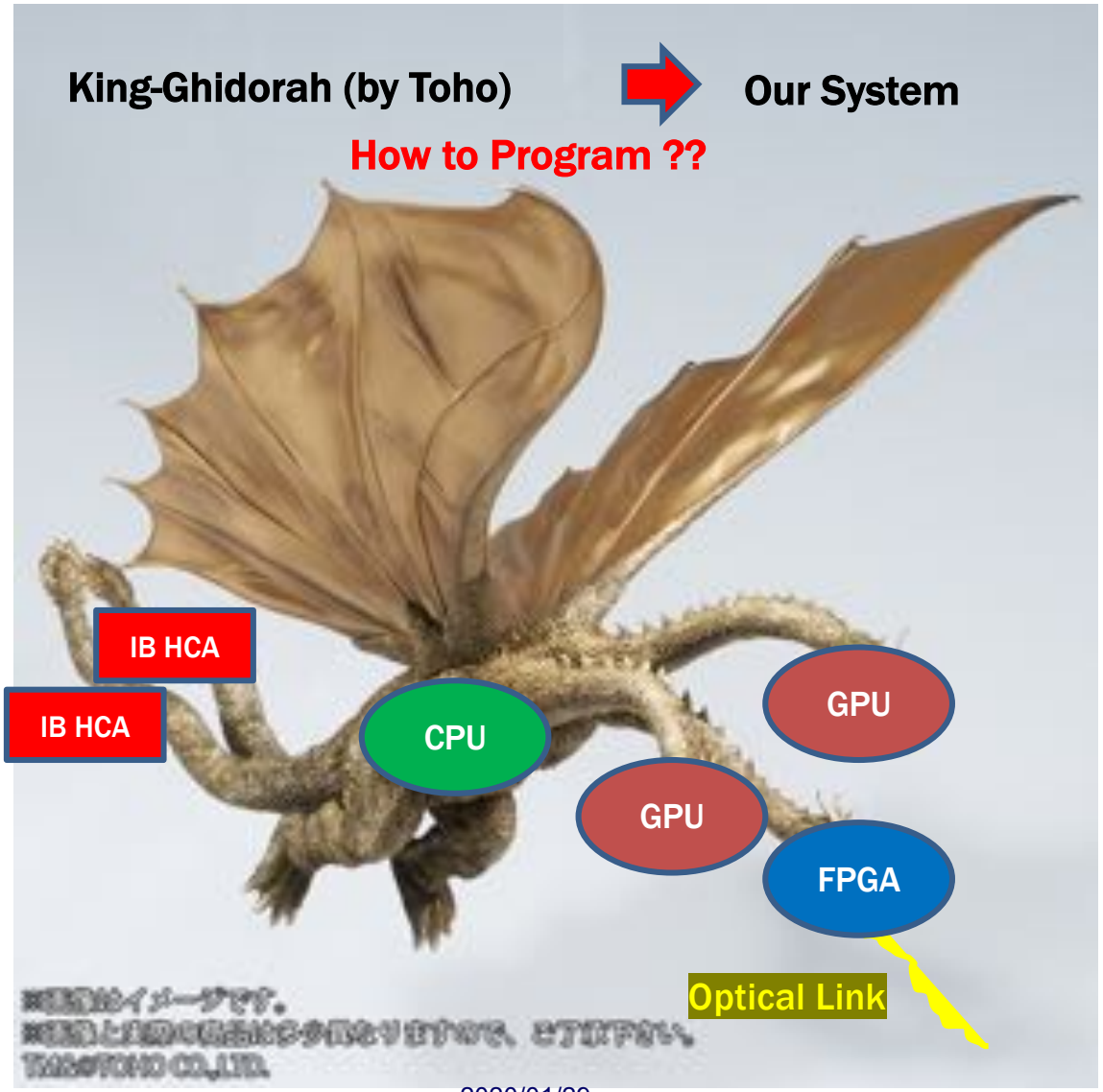
# Communication Bandwidth (on Arria10 – V100)




[Reference]

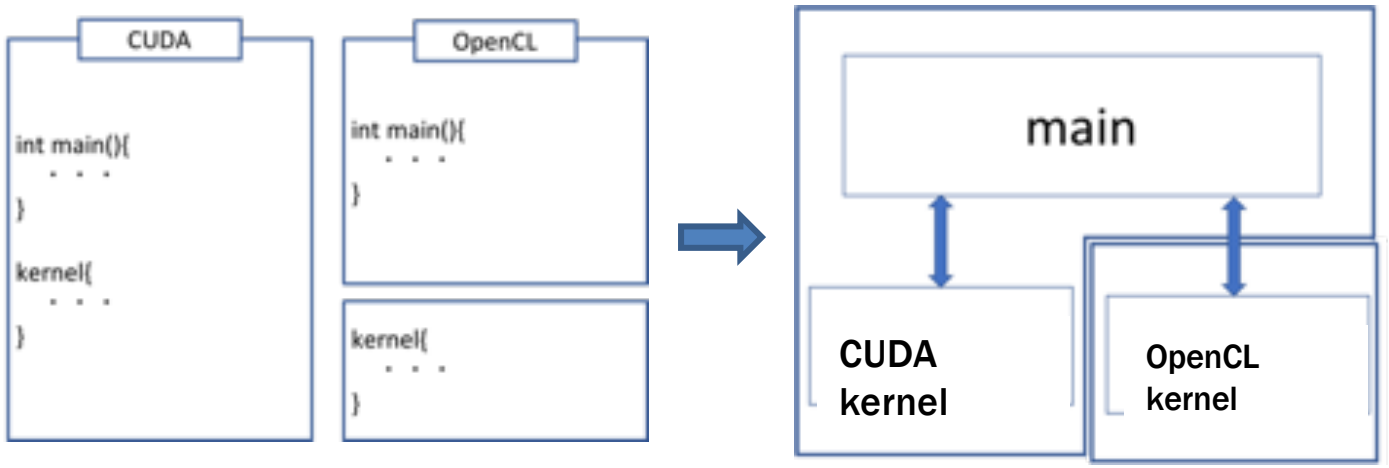
- Ryohei Kobayashi, Norihisa Fujita, Yoshiki Yamaguchi, Ayumi Nakamichi, Taisuke Boku, "GPU-FPGA Heterogeneous Computing with OpenCL-enabled Direct Memory Access", Proc. of Int. Workshop on Accelerators and Hybrid Exascale Systems (AsHES2019) in IPDPS2019 (to be published), May 20th, 2019.


King-Ghidorah (by Toho)


King-Ghidorah (by Toho)
Our System
How to Program ??
IB HCA
IB HCA
CPU
GPU
GPU
FPGA
Optical Link

# CUDA (GPU) + OpenCL (FPGA)

- **Calling two device Kernels written in CUDA (for GPU) and OpenCL (for FPGA)**
  - CUDA compiler (NVIDIA/PGI) and OpenCL compiler (Intel)
    → Two ”host” program exist
    Behavior of Host Program differs on two systems, but can be combined
    → One Host Program calls different system kernels
- We found the library to be resolved for each compiler and confirmed that hey don’t conflict
  → Linking everything

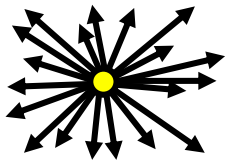


# Application Example – ARGOT (collab. with M. Umemura et. al.)

- ARGOT (Accelerated Radiative transfer on Grids using Oct-Tree)
  - Simulator for early stage universe where the first stars and galaxies were born
  - Radiative transfer code developed in Center for Computational Sciences (CCS), University of Tsukuba
  - CPU (OpenMP) and GPU (CUDA) implementations are available
  - Inter-node parallelisms is also supported using MPI
- ART (Authentic Radiation Transfer) method
  - It solves radiative transfer from light source spreading out in the space
  - Dominant computation part (90%~) of the ARGOT program
- In this research, we accelerate the ART method on an FPGA using Intel FPGA SDK for OpenCL as an HLS environment

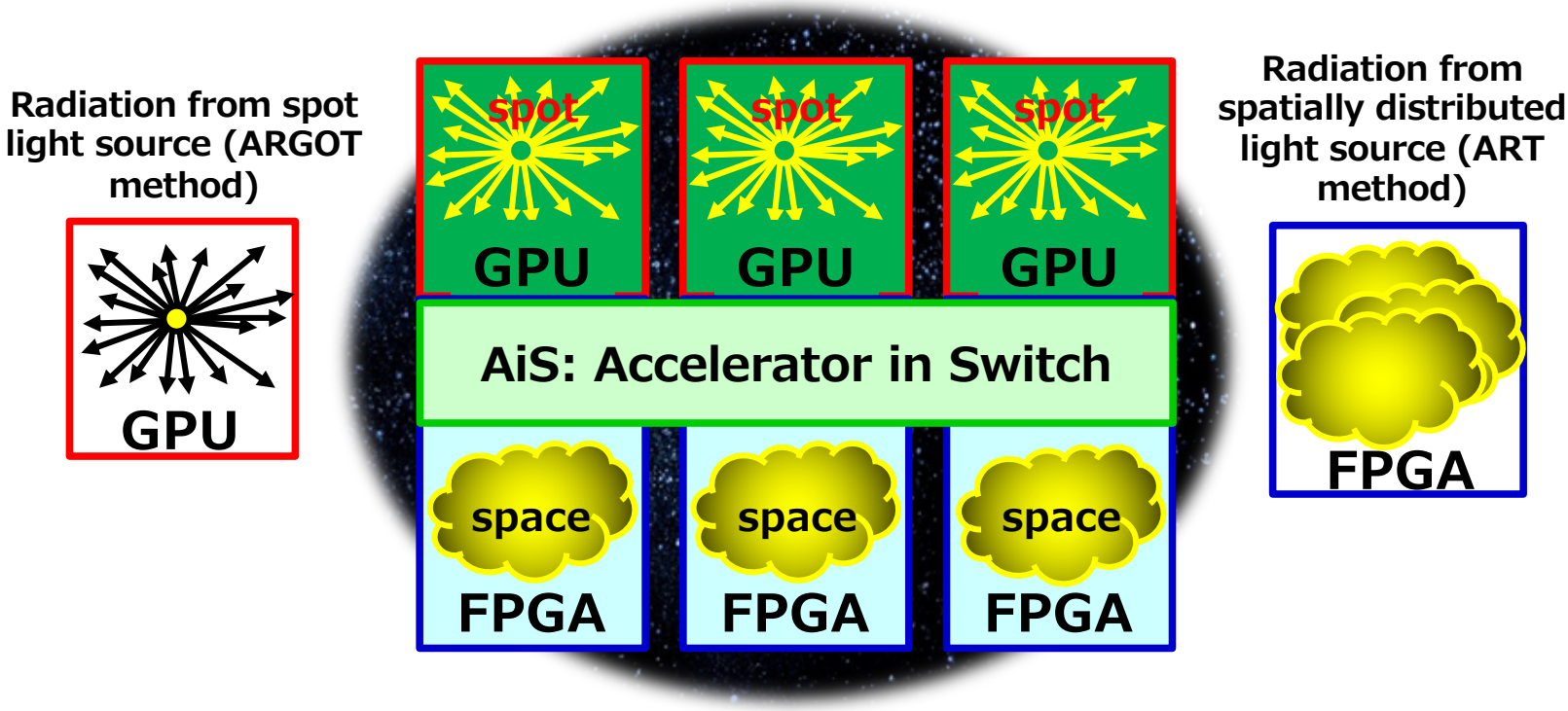
# ARGOT code: radiation transfer simulation

Radiation from spot light source (ARGOT method)



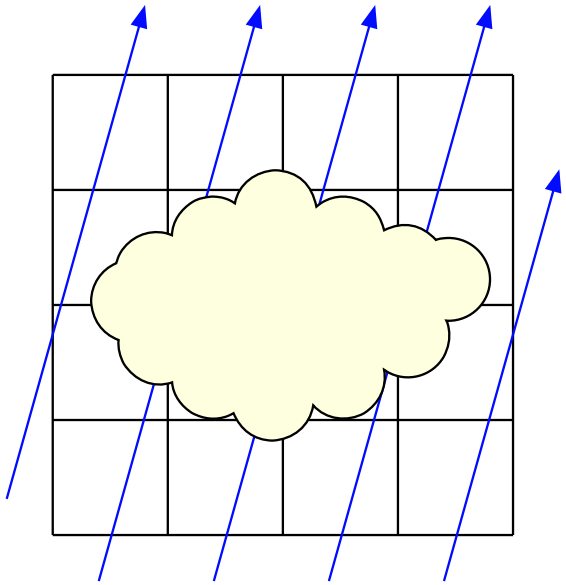
Radiation from spatially distributed light source (ART method)

# ARGOT code: radiation transfer simulation



# ART Method

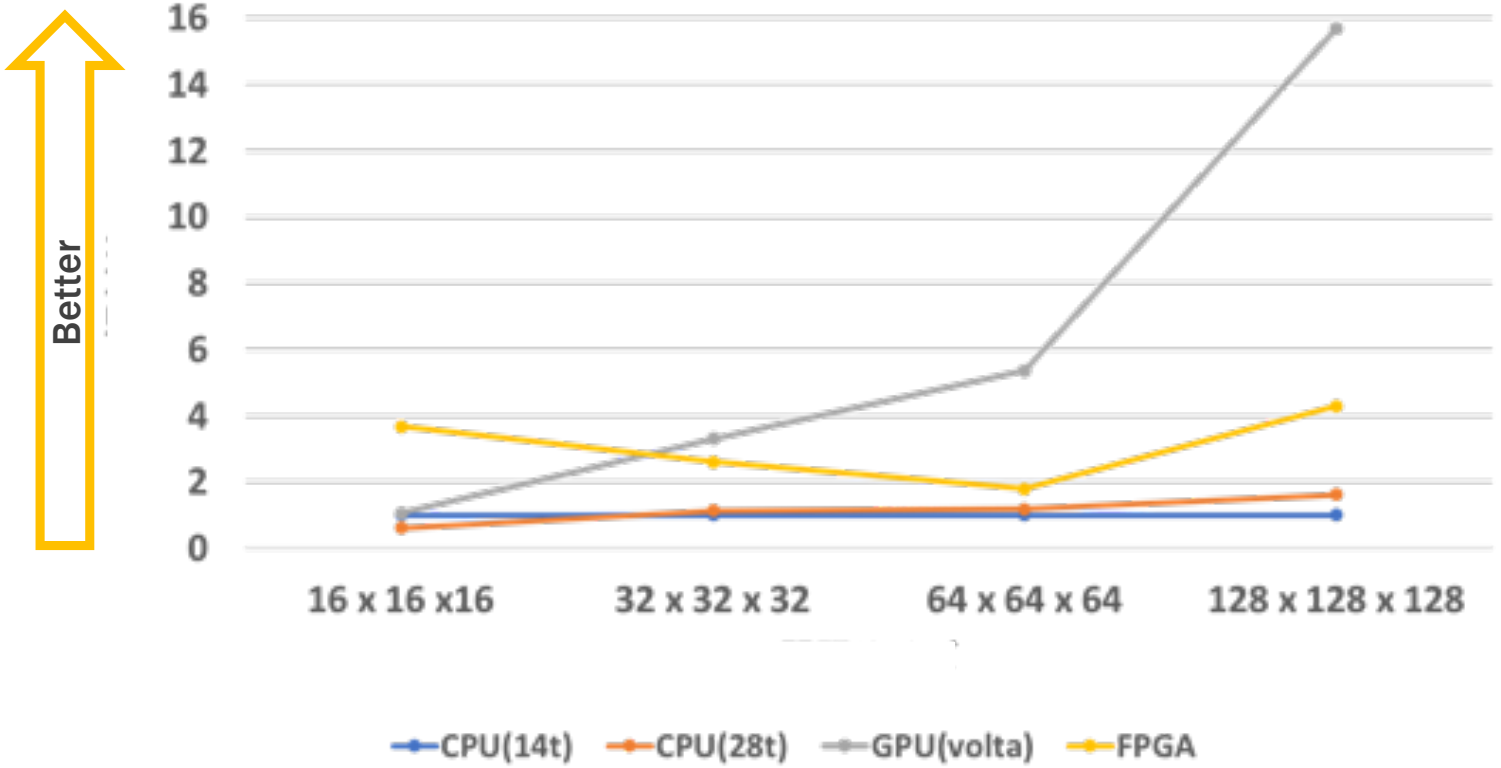
- ART method is based on ray-tracing method
  - 3D target space split into 3D meshes
  - Rays come from boundaries and move in straight in parallel with each other
  - Directions (angles) are given by HEALPix algorithm
- ART method computes radiative intensity on each mesh as shows as formula (1)
  - Bottleneck of this kernel is the exponential function (expf)
  - There is one expf call per frequency (v). Number of frequency is from 1 to 6 at maximum, depending on the target problem
  - All computation uses single precision computations
- Memory access pattern for mesh data is varies depending on ray's direction
  - Not suitable for SIMD style architecture
  - FPGAs can optimize it using custom memory access logics.



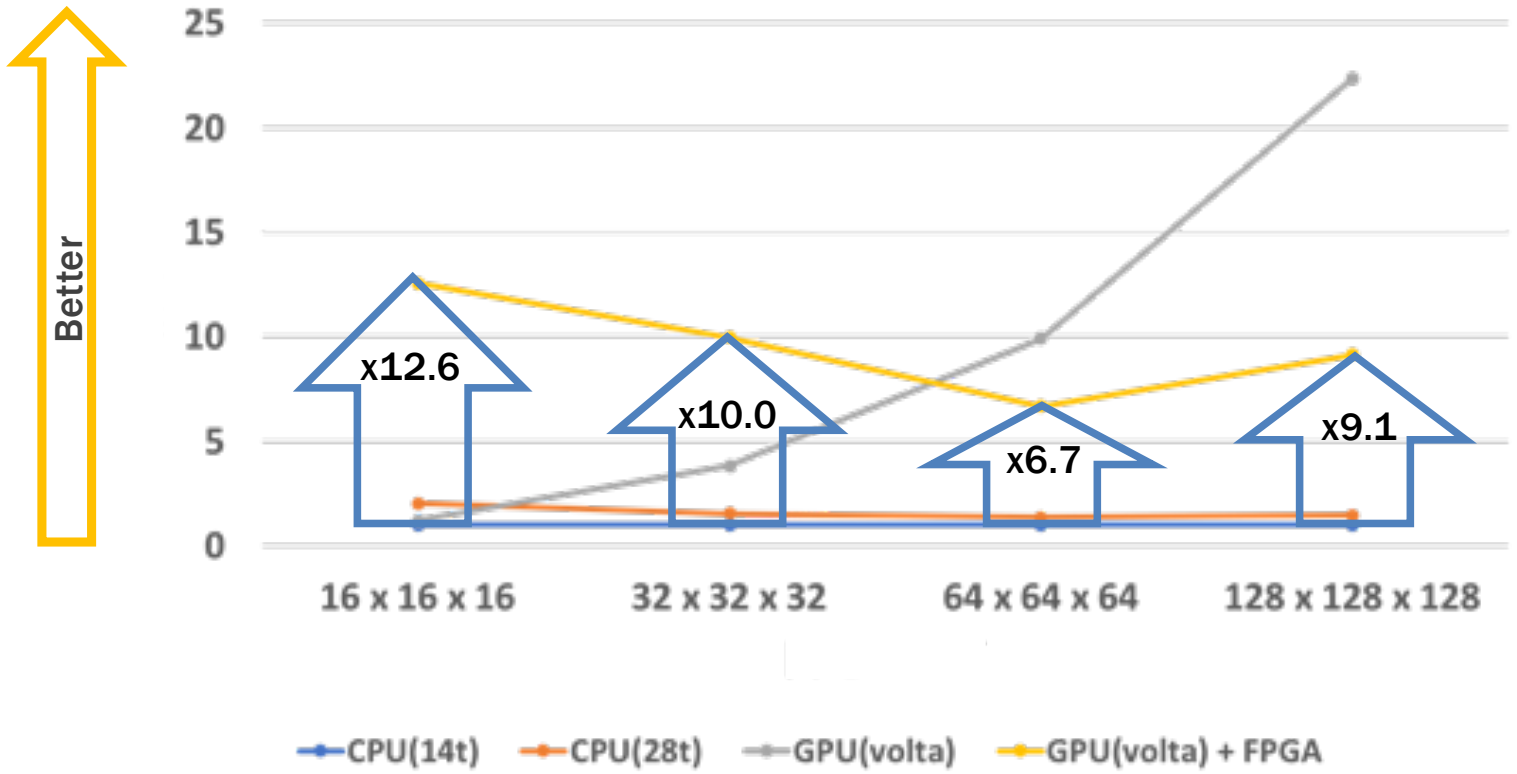
$$I_\nu^{out}(\hat{\boldsymbol{n}}) = I_\nu^{in}(\hat{\boldsymbol{n}})e^{-\Delta\tau_\nu} + S_\nu(1 - e^{-\Delta\tau_\nu}) \qquad (1)$$

# ART method on FPGA (on Intel Arria10)

- OpenCL-base ART method by N. Fujita [HEART2018]
- Performance improvement from Xeon CPU (14 core & 28 core)



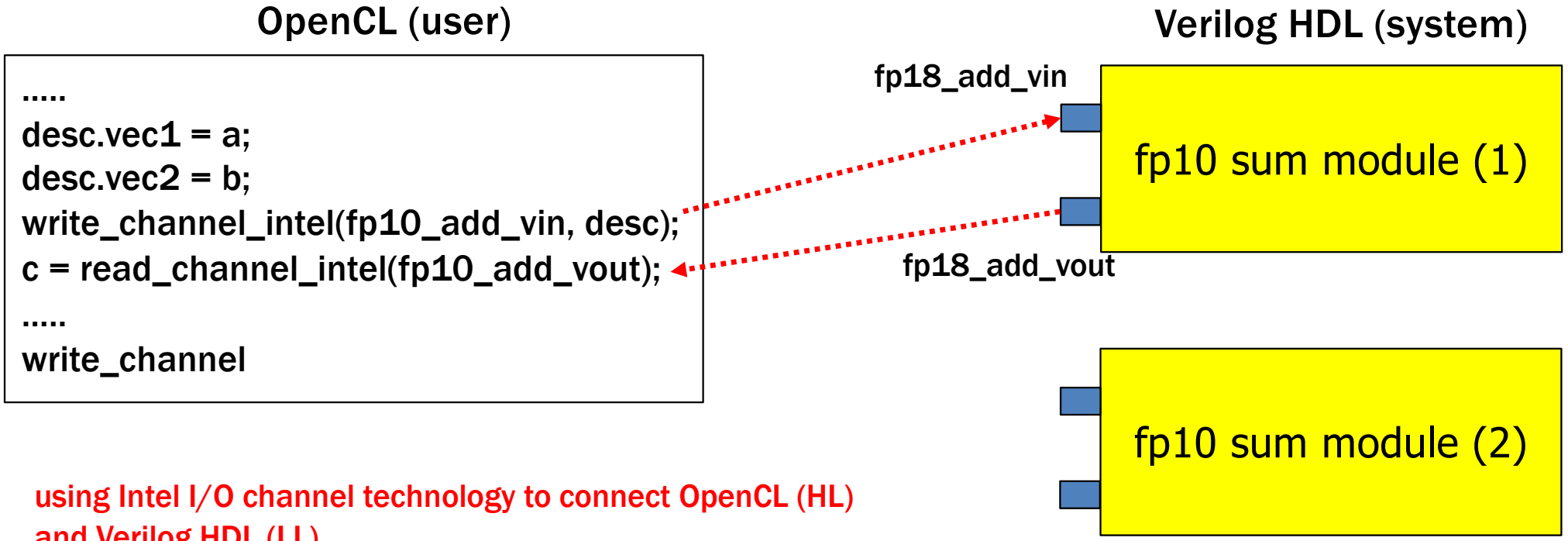
# ARGOT program full code with GPU+FPGA



# How to use “weird” precision calculation

- Ideal solution is to make a special compiler to handle non-standard FP data, but it takes a time...
- We need to program it in Verilog HDL (very low level HDL)
- It is possible to be called from OpenCL code by Intel I/O channel feature, in the same manner with library call
- What kind of weird-precision math routine you need ?
- Codesigning of application-mathematics-engineering
- For some part of applications which still need standard SP or DP, we can use GPU on Cygnus because it is the world first supercomputer with both accelerators

# Coupling with OpenCL (idea)




- using Intel I/O channel technology to connect OpenCL (HL) and Verilog HDL (LL)
- for parallel pipelining, functions should be duplicated (fp18_add_vin1, fp18_add_vin2, ...)
- various type of function modules should be prepared in Verilog HDL module

# Summary


- Cygnus is a multi-hybrid accelerated supercomputer based on AiS (Accelerator in Switch) concept
- Cygnus is equipped with very high performance GPU and FPGA (partially) to make “strong scaling ready” accelerated system for applications where GPU-only solutions are weak, as well as all kind of GPU-ready applications
- FPGA for HPC is a new concept toward next generation’s flexible and low power solution beyond GPU-only computing
- Multi-physics simulation is the first stage target of Cygnus and will be expanded to variety of applications where GPU-only solution has some bottleneck
- Algorithms and applications with mixed- or minimal- precision computation is a big challenge, and the flexibility of FPGA will greatly help it
- Our collaboration will provide the supporting modules and OpenCL-ready interface as well as numerical algorithm development