Multi-Hybrid Device Programming and Application by Uniform Language

Taisuke Boku

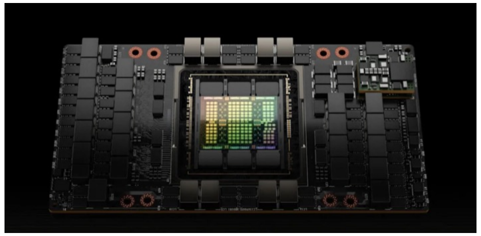
Director, Center for Computational Sciences

University of Tsukuba

taisuke@ccs.tsukuba.ac.jp

Accelerators in HPC ⇒ majority = GPU

- 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 divergence)
    - frequent inter-node communication (kernel switch, go back to CPU)



NVIDIA Tesla H100
Hopper Architecture GPU
(from NVIDIA web page)

FPGA in HPC

- Goodness of recent FPGA for HPC
  - True codesigning with applications (essential)
  - Programmability improvement: OpenCL, other high level languages
  - High performance interconnect: 100Gbps x 4
  - 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, but still difficult to use

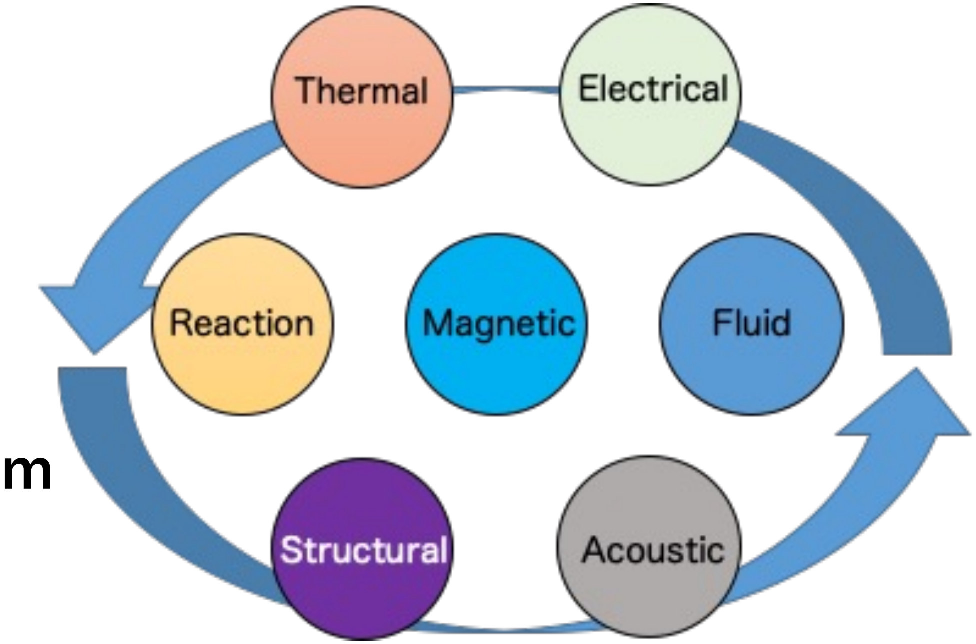
BittWare 520N with Intel Stratix10 FPGA equipped with 4x 100Gbps optical interconnection interfaces

GPU vs FPGA as HPC solutions

device	GPU	FPGA
parallelization	SIMD (x multi-group)	pipeline (x multi-group)
standard FLOPS	😃😃 (>1000 cores)	😃 (~100 pipeline)
conditional branch	😢 (warp divergence)	😃 (both direction)
memory	😃😃 (HBM2e)	😢 (DDR)⟶😃 (HBM2)
interconnect	😢 (via host facility)	😃😃 (own optical links)
programming	😃 (CUDA, OpenACC, OpenMP)	😢 (HDL)⟶😏 (HLS)
self-controllability	😢 (slave device of host CPU)	😃 (autonomous)
HPC applications	😃 (various fields)	😢 (limited)

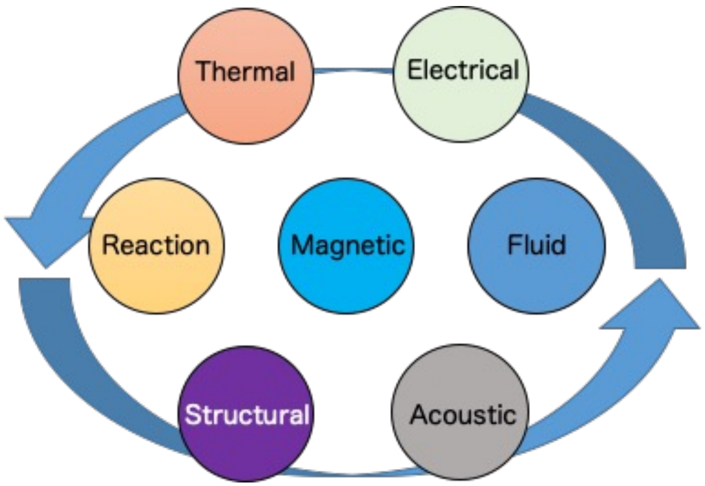
Coupling GPU and FPGA, why ?

- Many multi-physical simulation to combine several sorts of different phenomena on a system is required in advanced physics
  - space – particle reaction
  - fluid dynamics with chemical reaction
  - macroscopic/microscopic hybrid simulation molecular simulation
- Characteristics and dynamism of parallelism drastically changes during the simulation
  - SIMD friendly or pipeline friendly
  - small fraction of low degree parallelism in a code makes “last one mile problem” under Amdahl‘s Law

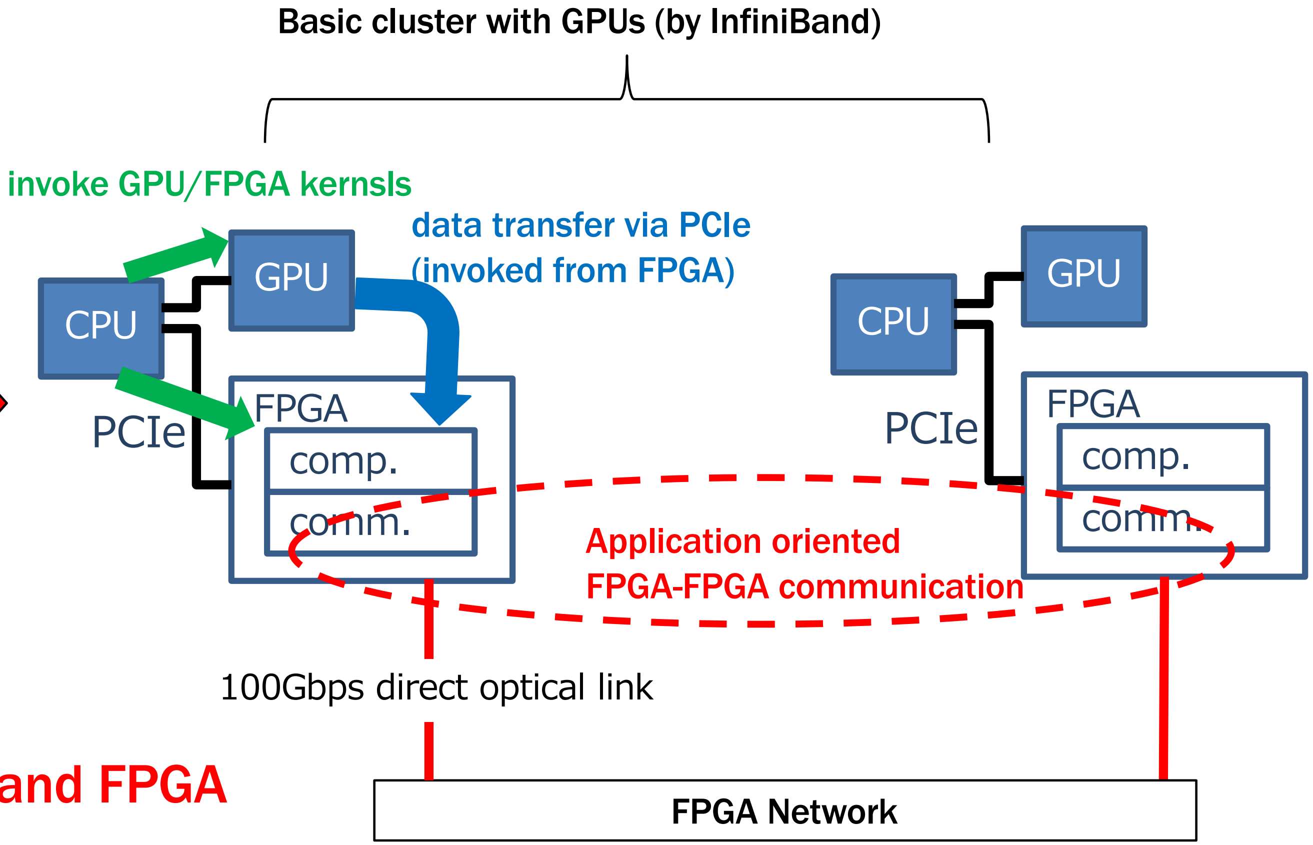


CHARM: Cooperative Heterogeneous Acceleration with Reconfigurable Multi-devices

multi-physics/multi-scale
complicated problem



Thermal
Electrical
Reaction
Magnetic
Fluid
Structural
Acoustic



Basic cluster with GPUs (by InfiniBand)

invoke GPU/FPGA kernsls

data transfer via PCIe
(invoked from FPGA)

CPU
GPU
PCIe
FPGA
comp.
comm.

CPU
GPU
PCIe
FPGA
comp.
comm.

Application oriented
FPGA-FPGA communication

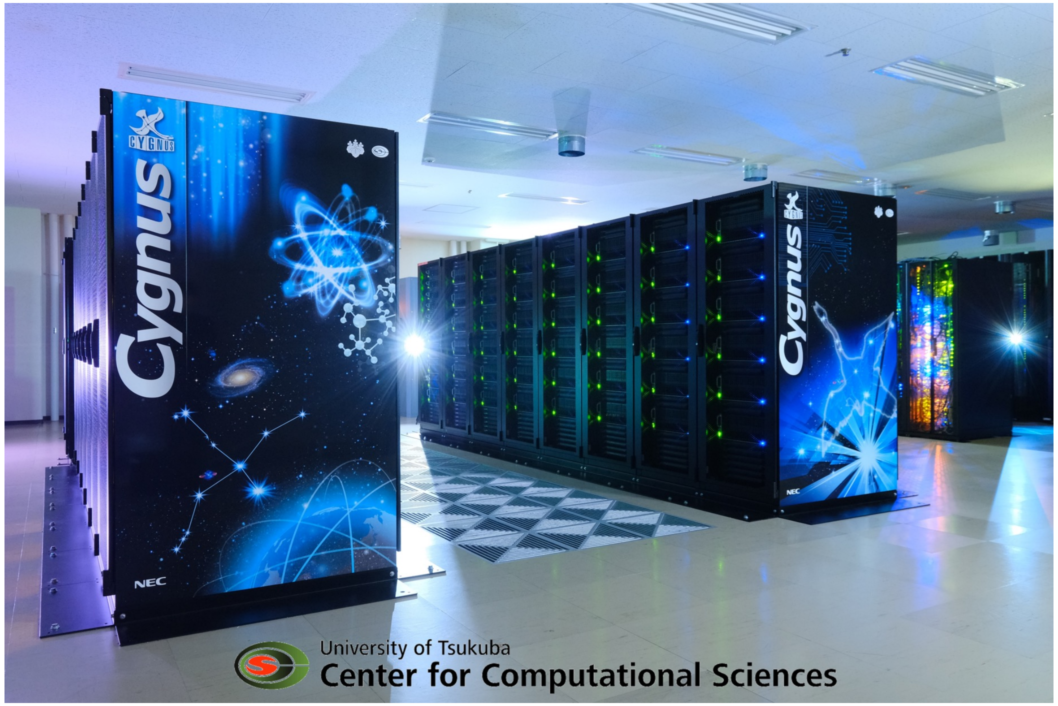
100Gbps direct optical link

FPGA Network

Cooperative computing with GPU and FPGA

Cygnus: world first multi-hybrid cluster with GPU+FPGA

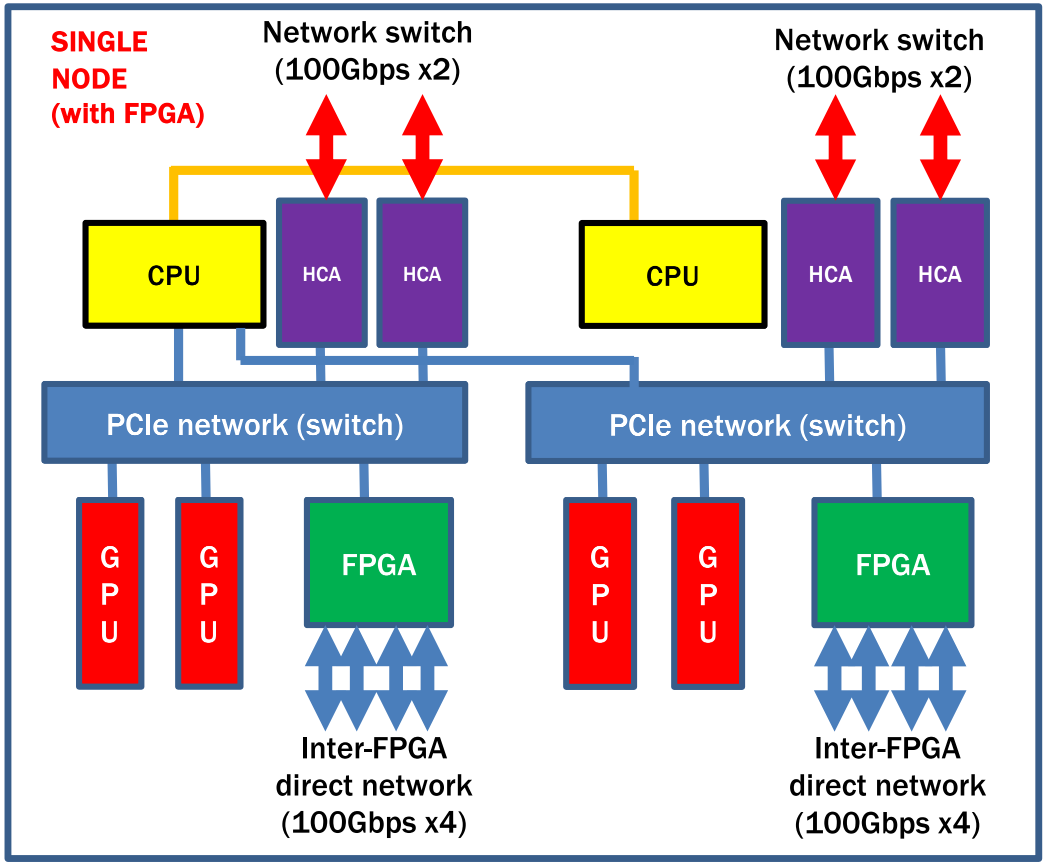
@ CCS, Univ. of Tsukuba (deployed by NEC)



Cygnus supercomputer at Center for Computational Sciences, Univ. of Tsukuba (Apr. 2019~)
85 nodes in total including 32 “Albireo” nodes with GPU+FPGA (other “Deneb” nodes have GPU only)

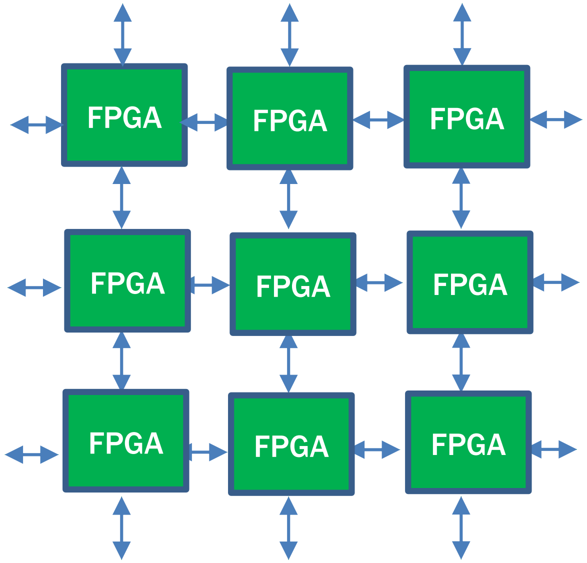
Single node configuration (Albireo)

- 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


SINGLE NODE (with FPGA)
Network switch (100Gbps x2)
Network switch (100Gbps x2)
CPU
HCA
HCA
CPU
HCA
HCA
PCIe network (switch)
PCIe network (switch)
GPU
GPU
FPGA
GPU
GPU
FPGA
Inter-FPGA direct network (100Gbps x4)
Inter-FPGA direct network (100Gbps x4)

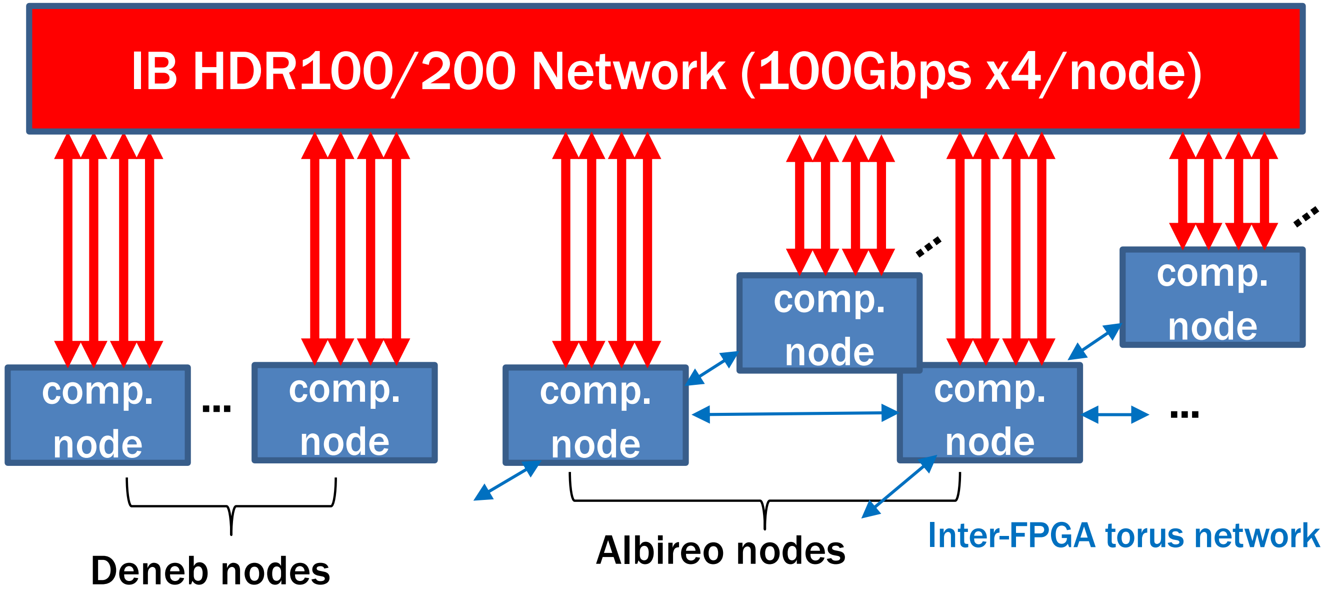
Two types of interconnection network

Inter-FPGA direct network (only for Albireo nodes)

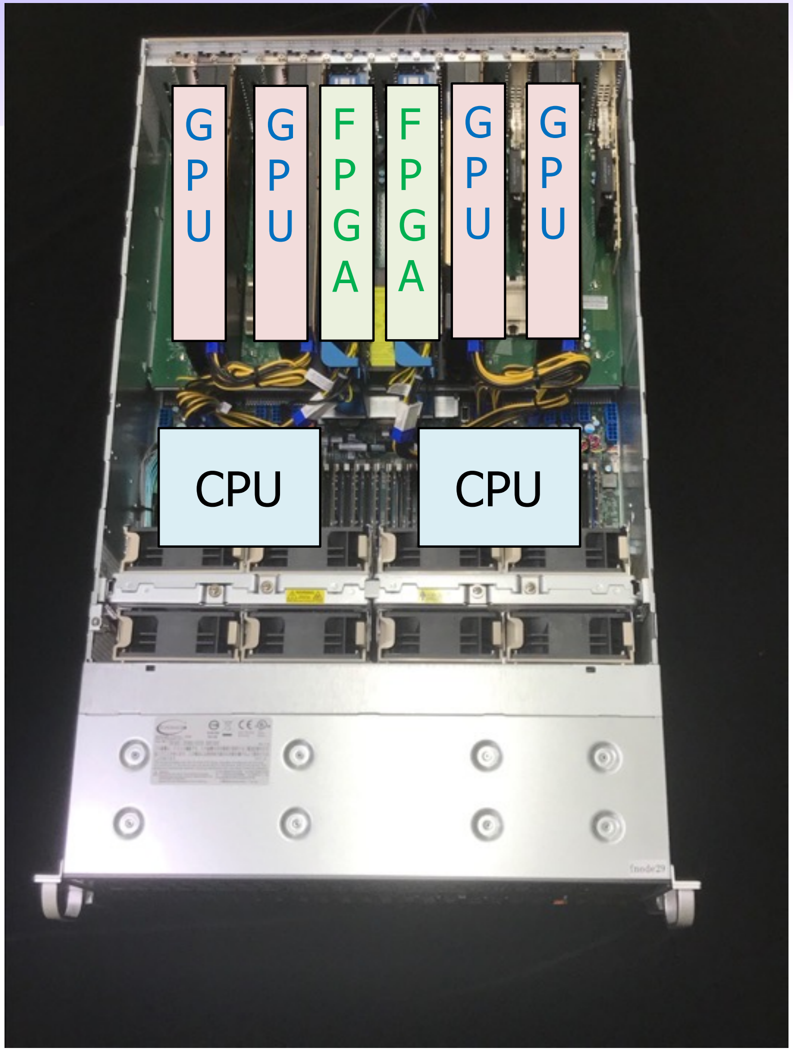


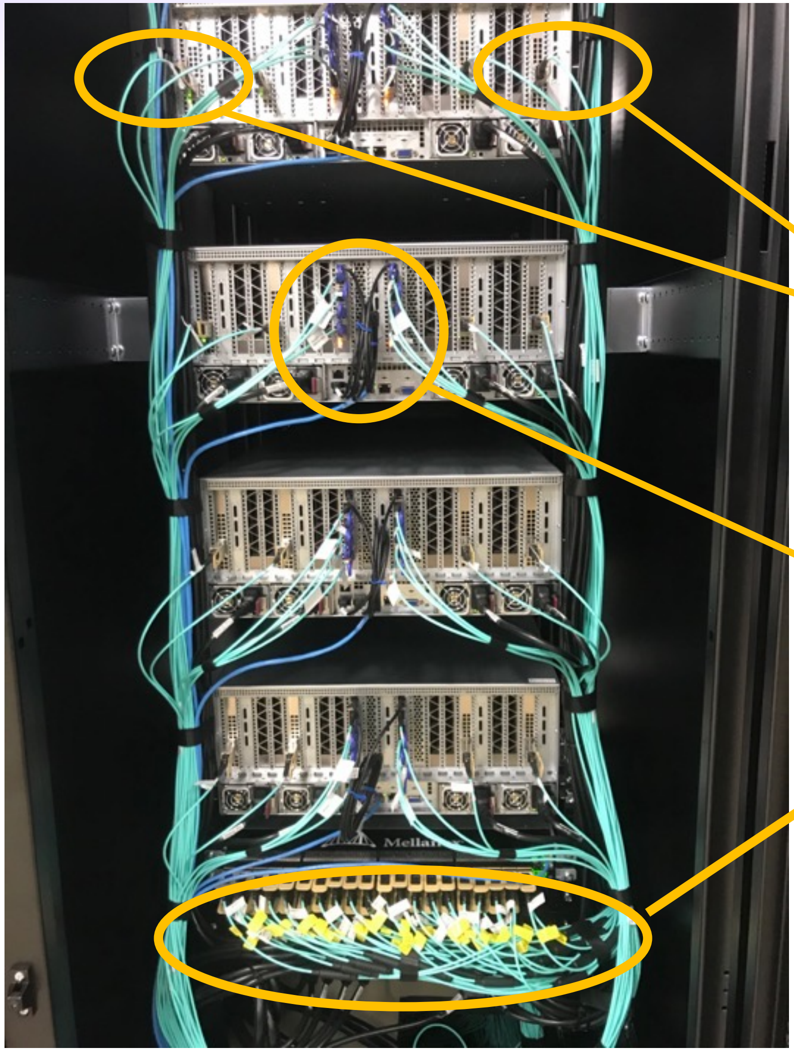
64 of FPGAs on Albireo nodes (2FPGAS/node) are connected by 8x8 2D torus network without switch

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 x2

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

1.2Tbps/node

Albireo node

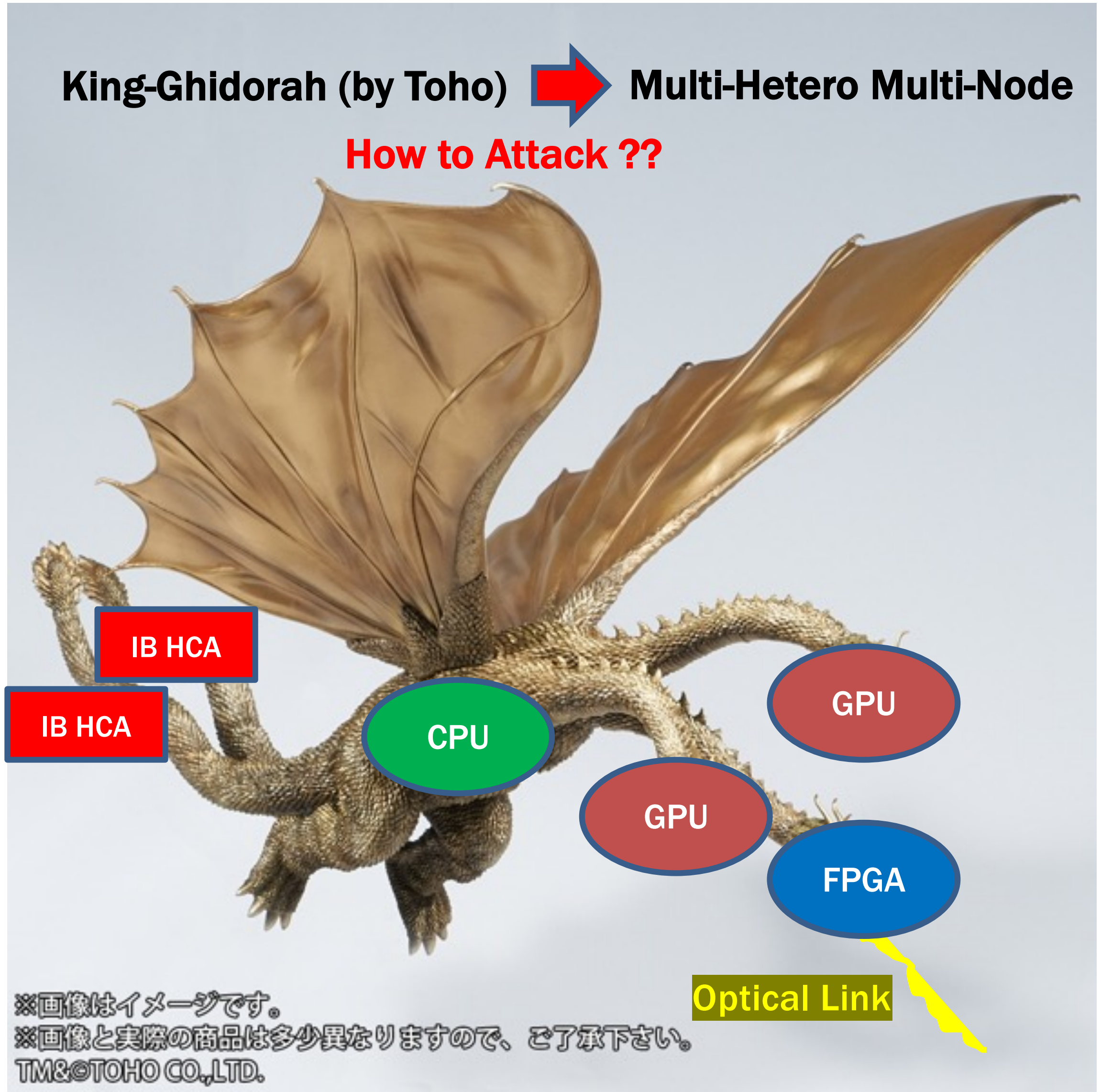
King-Ghidorah (by Toho)

※画像はイメージです。

※画像と実際の商品は多少異なりますので、ご了承下さい。

King-Ghidorah (by Toho) ➡ Multi-Hetero Multi-Node

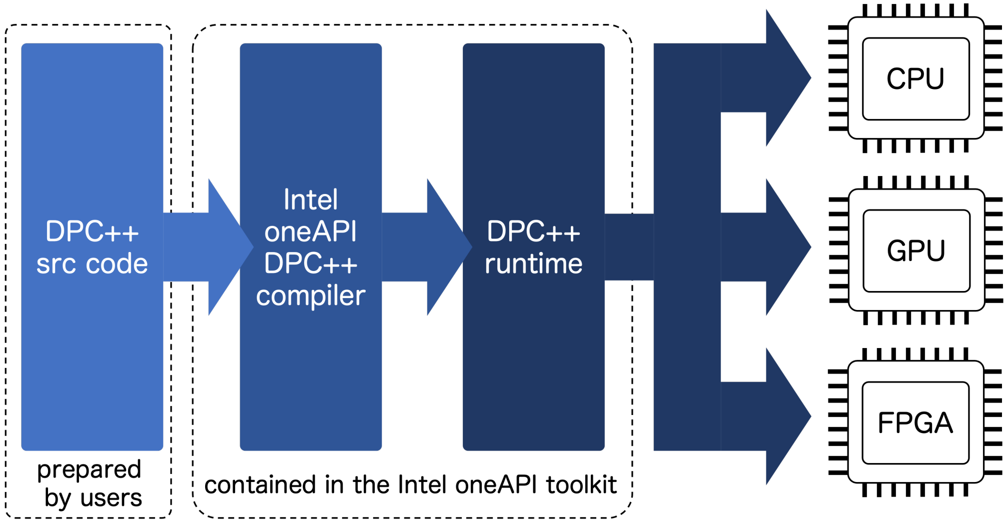
How to Attack ??



※画像はイメージです。
※画像と実際の商品は多少異なりますので、ご了承下さい。
TM&©TOHO CO.,LTD.

Possible coding method: oneAPI

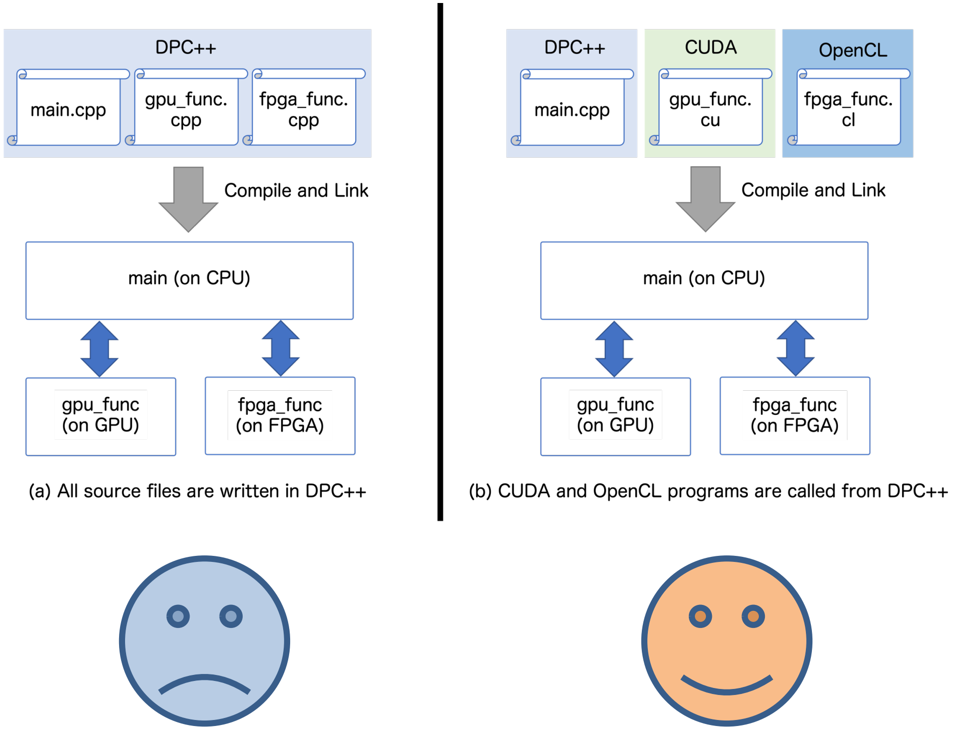
- oneAPI is a cross-architecture programming framework
  - simplify the development across different architectures
- Data Pallalell C++ (DPC++)
  - DPC++ = C++ + SYCL + extensions
  - enables unified description across different architectures



CHARM by oneAPI

- In oneAPI, programming in DPC++ is recommended
  - (a) approach
  - Problem: Existing GPU and FPGA code is written in other languages such as CUDA, OpenCL, etc

    These code assets already exist
  - Reimplementation by DPC++ is a burden for users
- oneAPI also can use modules written in other languages
  - (b) approach
  - Code can be reused



DPC++
main.cpp
gpu_func.cpp
fpga_func.cpp
Compile and Link
main (on CPU)
gpu_func (on GPU)
fpga_func (on FPGA)

(a) All source files are written in DPC++

DPC++
main.cpp
CUDA
gpu_func.cu
OpenCL
fpga_func.cl
Compile and Link
main (on CPU)
gpu_func (on GPU)
fpga_func (on FPGA)

(b) CUDA and OpenCL programs are called from DPC++

How much easier to program using oneAPI?

CUDA + OpenCL programming model

CUDA API

```
cudaMemcpy(...);
// Call the CUDA kernel
gpu_kernel<<<grid, block>>>(...);
cudaMemcpy(...);
```

OpenCL API

```
clEnqueueWriteBuffer(...);
clSetKernelArg(...);
// Call the OpenCL kernel
clEnqueueTask(..., fpga_kernel,...);
clEnqueueReadBuffer(...);
```

oneAPI calling CUDA+OpenCL kernels

```
gpu_queue.memcpy(...);
gpu_queue.submit([&](handler& h) {
   h.interop_task([=](interop_handler ih) {
     // Call the CUDA kernel
     gpu_kernel<<<grid, block>>>(...);
   });
});
gpu_queue.memcpy(...);
```

```
fpga_queue.memcpy(...);
fpga_queue.submit([&](sycl::handler &h) {
       // Call the OpenCL kernel
       h.set_args(...);
       h.single_task(fpga_kernel);
});
fpga_queue.memcpy(...);
```

using oneAPI, we can control GPU and FPGA in the same style ⇨ **boring!!**

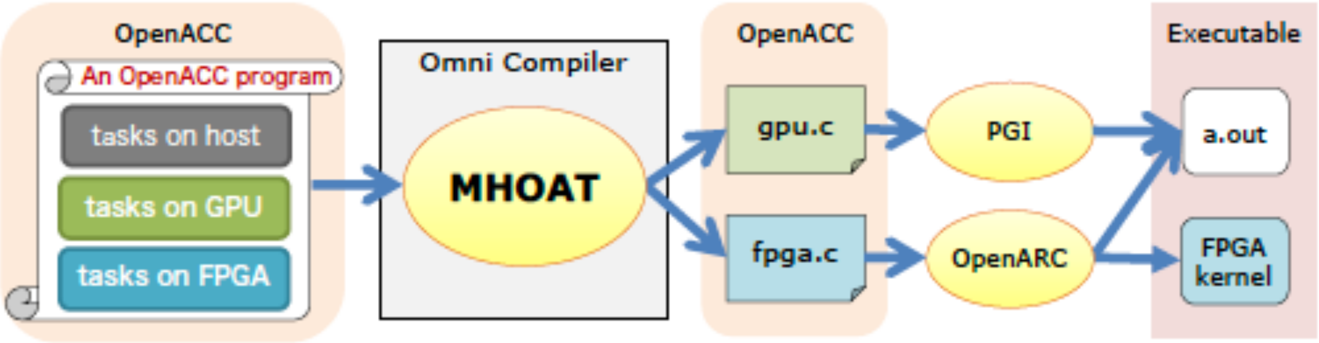
New method: Unified OpenACC meta-compiler

- OpenACC is available both for GPU and FPGA
  - under development in collaboration between CCS-Tsukuba and FTG-ORNL
  - GPU compilation – PGI OpenACC compiler
  - FPGA compilation – OpenARC for FPGA on OpenACC, developed at ORNL
    -> collaboration between CCS and ORNL (Jeff Vetter, Seyong Lee, etc.)
- Solving some conflict on host code environment and run-time
  - Basic running is confirmed and testing example codes
- Issues
  - Performance enhancement on GPU and FPGA totally differs
    - GPU – horizontal (data) parallelism in SIMD manner
    - FPGA – clock level pipelining
    - Memory model difference: HBM2 vs BRAM

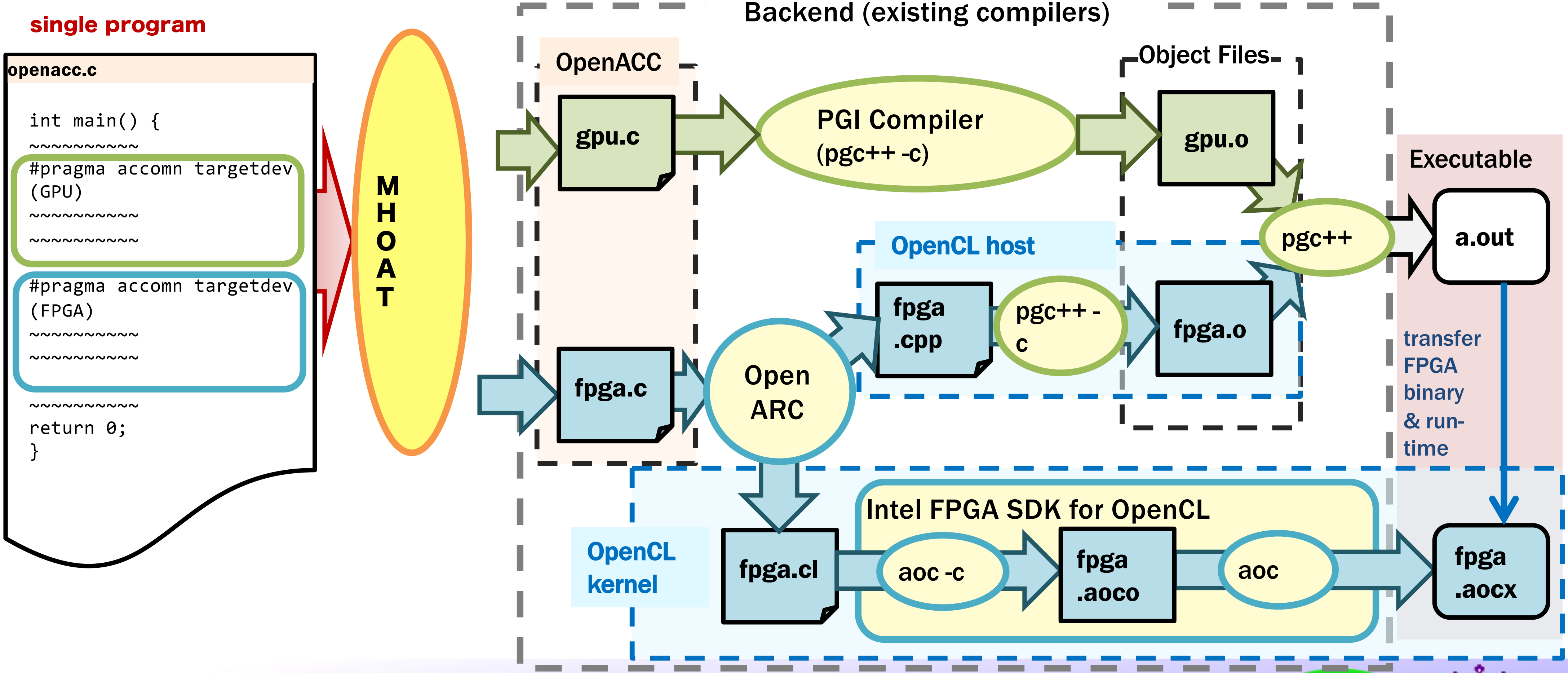
⇒ **MHOAT (Multi-Hybrid OpenACC Translator)**

MHOAT (Multi-Hybrid OpenACC Translator)

- User describes the code in OpenACC (currently C only)
  - considering the part to be offloaded to certain type of accelerator (GPU or FPGA)
    ⇨ introducing extended directive
    #pragma accomn target_dev(*device*)
  - data directive is treated according to the mapped device
- MHOAT separates the code block (task) to "host", "GPU" and "FPGA" execution
- Each block is compiled by back-end, then assembled for single executable and bit-stream for FPGA



Compilation flow of MHOAT



```
int main() {
~~~~~~~~~~
#pragma accomn targetdev
(GPU)
~~~~~~~~~~
~~~~~~~~~~
#pragma accomn targetdev
(FPGA)
~~~~~~~~~~
~~~~~~~~~~
~~~~~~~~~~
return 0;
}
```

Application Example – ARGOT code

- 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 code

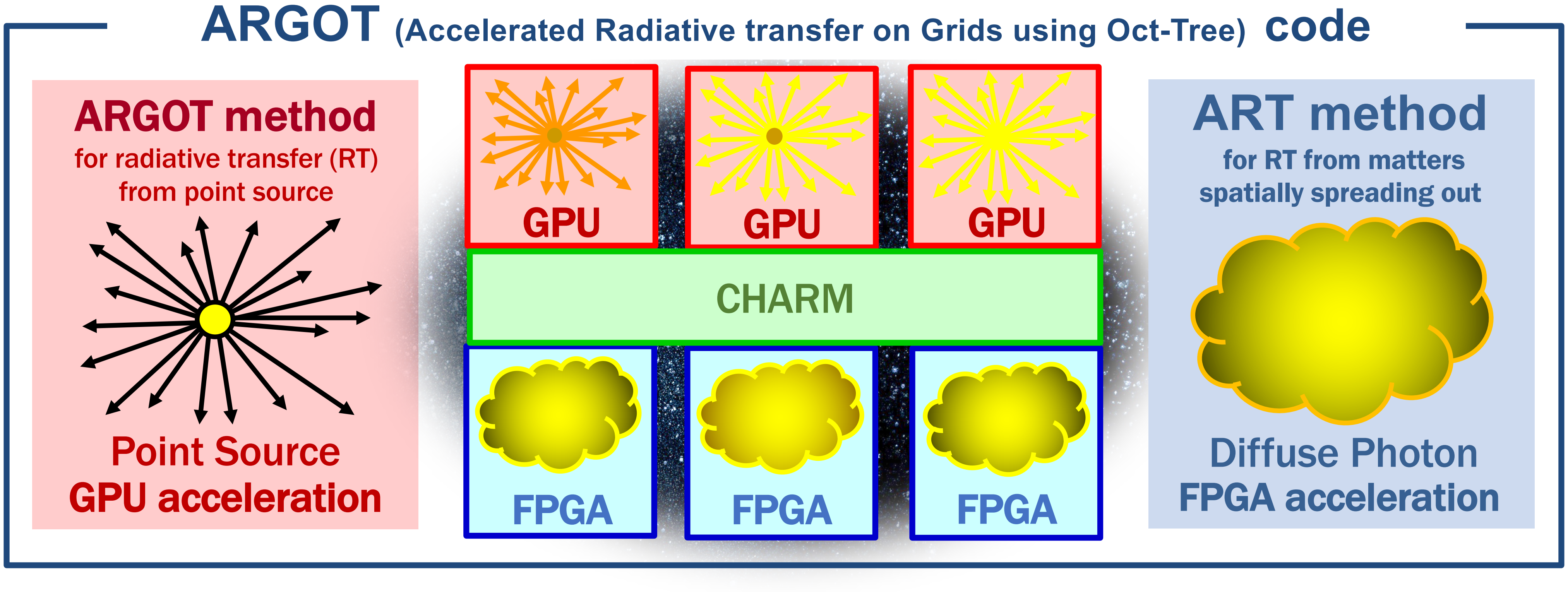
Two computation elements in ARGOT code: ARGOT method and ART method

- ARGOT method: Point Source processing
- ART method (Authentic Radiation Transfer): Diffused Photon processing

Point Source
Diffuse Photon

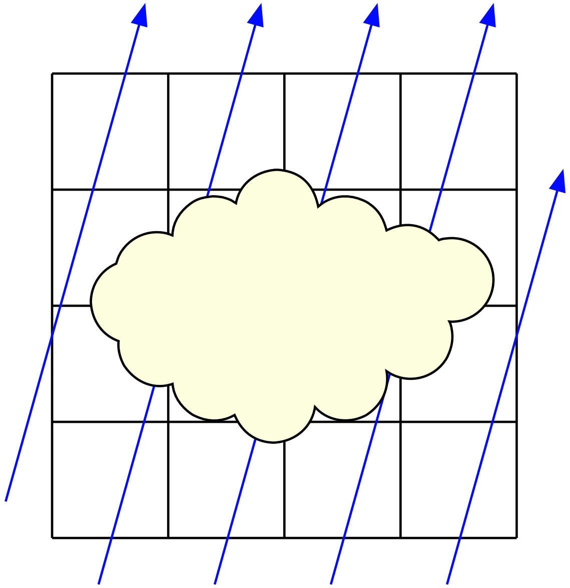
Two computation elements in ARGOT code: ARGOT method and ART method

- ARGOT method: Point Source processing
- ART method (Authentic Radiation Transfer): Diffused Photon processing



ART Method

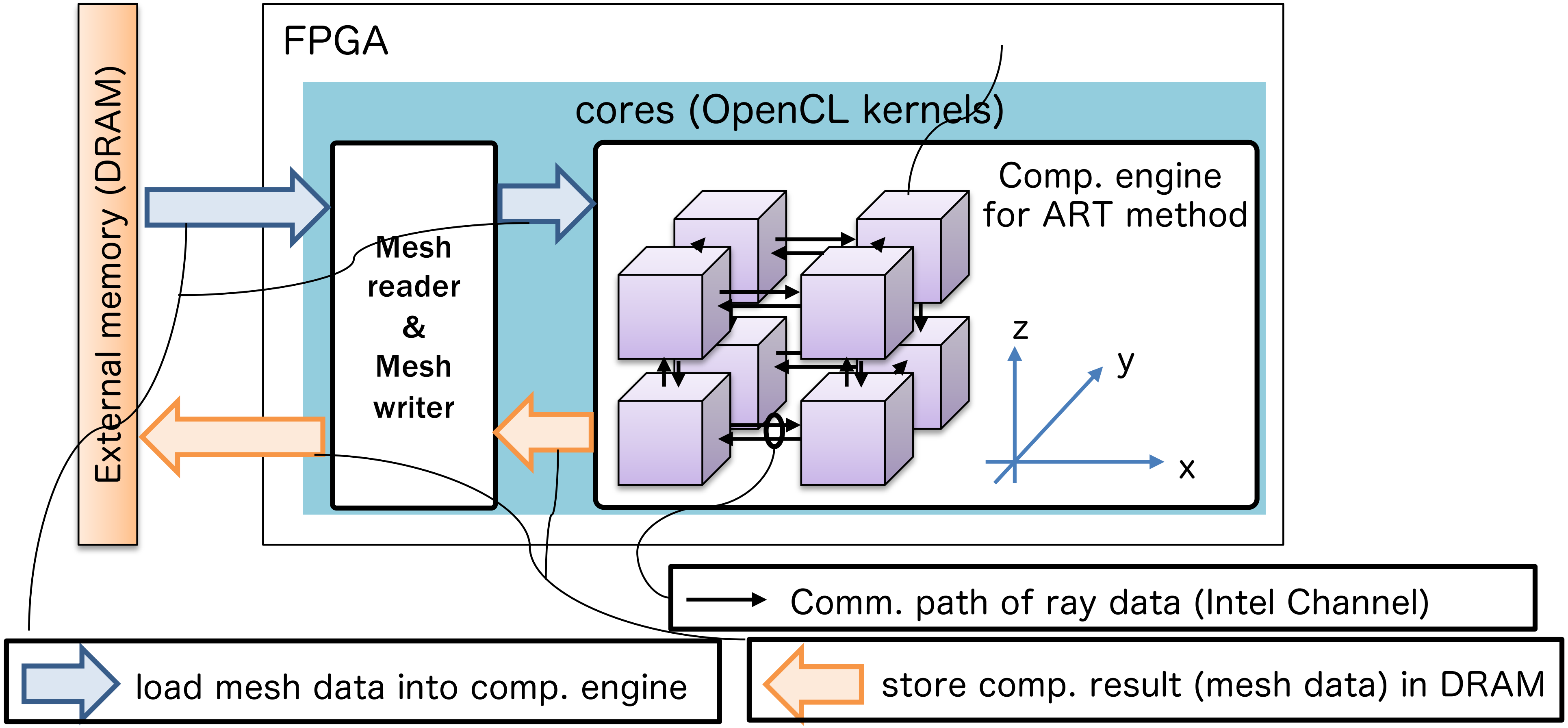
- ART method is based on ray-tracing method
  - 3D target space split into 3D meshes
  - Rays come from boundaries and move in straight in parallel with each other
  - Directions (angles) are given by HEALPix algorithm
- ART method computes radiative intensity on each mesh as shows as formula (1)
  - 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 BRAM (Block RAM = SRAM) and DRAM



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

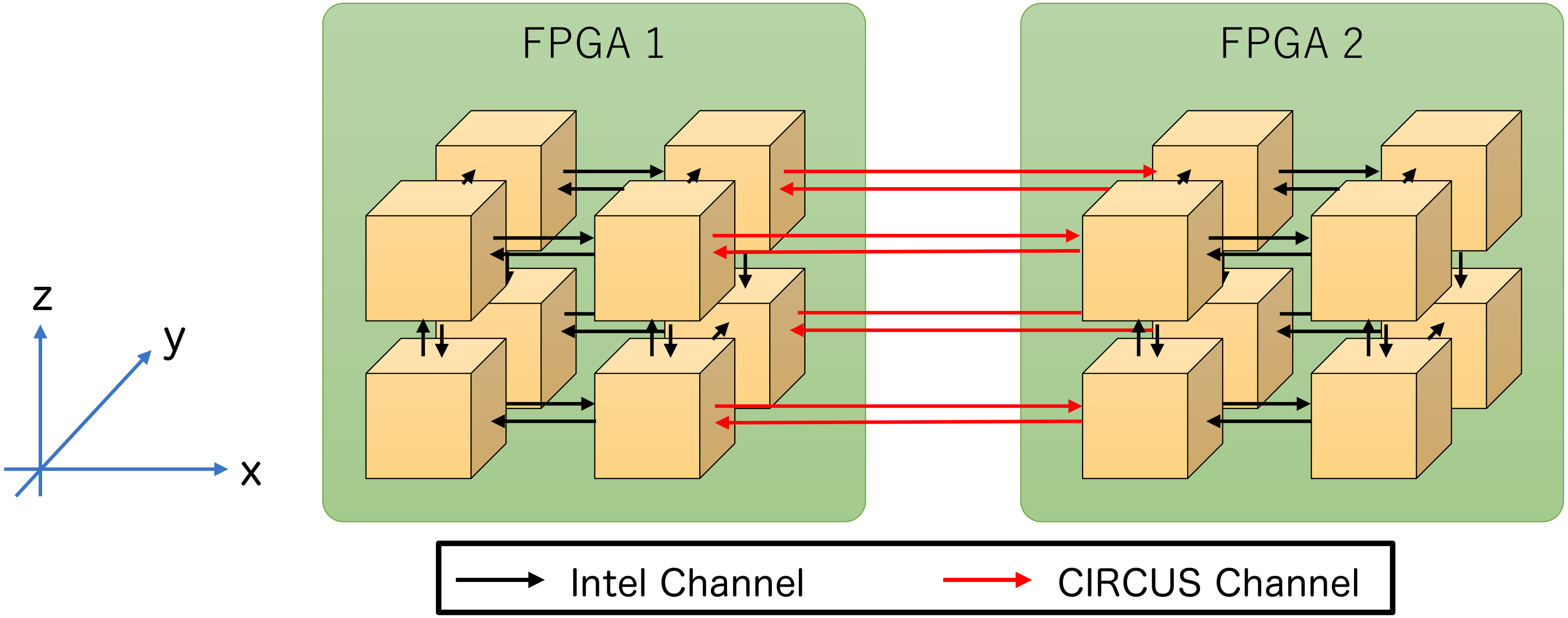
FPGA implementation of ART method

- **Domain Decomposition + Intel Channel for multi-FPGA expansion**

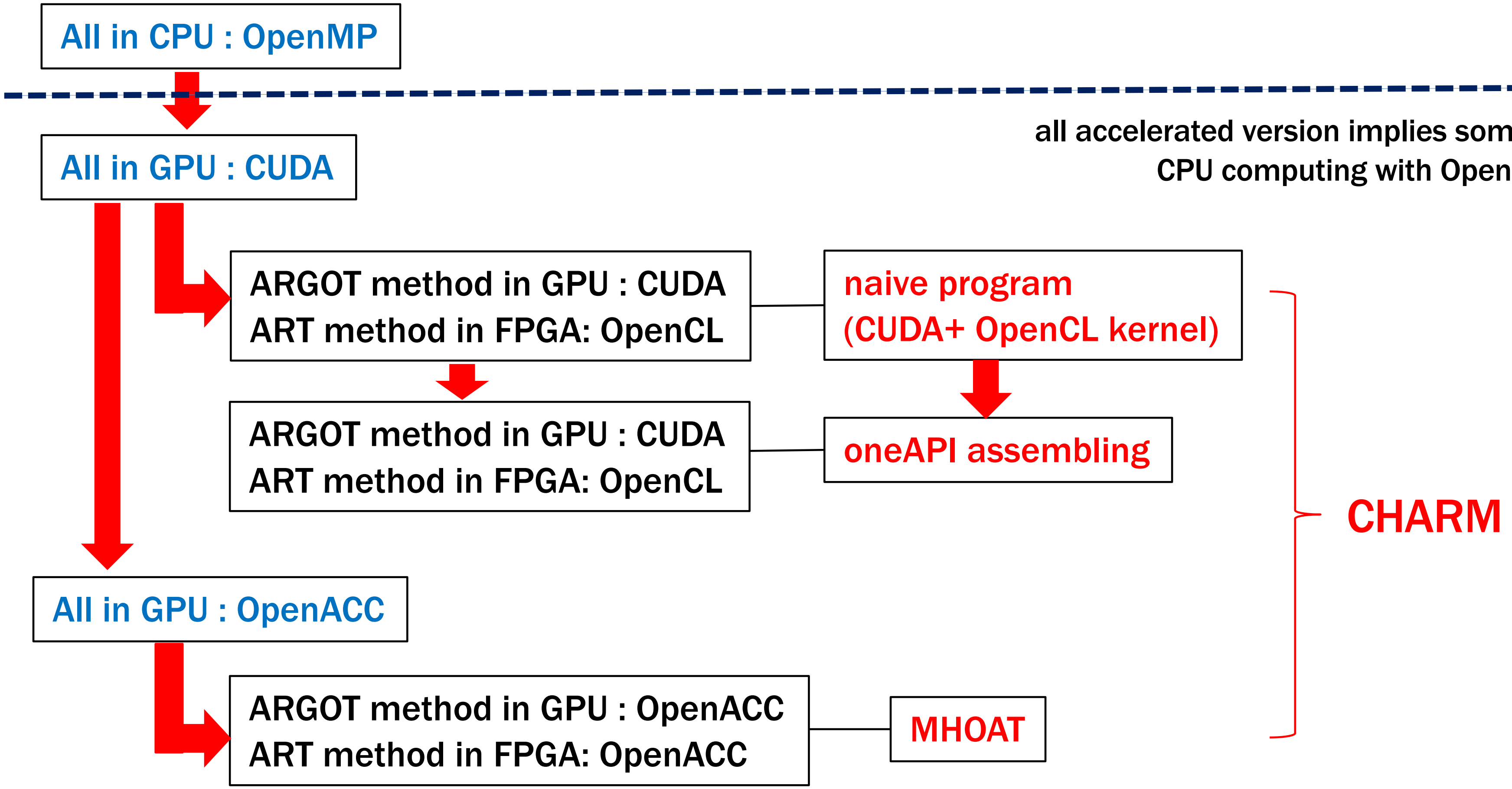

External memory (DRAM)
FPGA
cores (OpenCL kernels)
Mesh reader & Mesh writer
Comp. engine for ART method
z
y
x
Comm. path of ray data (Intel Channel)
load mesh data into comp. engine
store comp. result (mesh data) in DRAM

Parallel ART with multiple FPGAs

- replacing Intel Channel at FPGA border with CIRCUS
  - making cluster of ART execution kernels with Intel Channel or CIRCUS



Version history of ARGOT code



All in CPU : OpenMP

all accelerated version implies some fraction of CPU computing with OpenMP

All in GPU : CUDA

ARGOT method in GPU : CUDA
ART method in FPGA: OpenCL

naive program
(CUDA+ OpenCL kernel)

ARGOT method in GPU : CUDA
ART method in FPGA: OpenCL

oneAPI assembling

CHARM

All in GPU : OpenACC

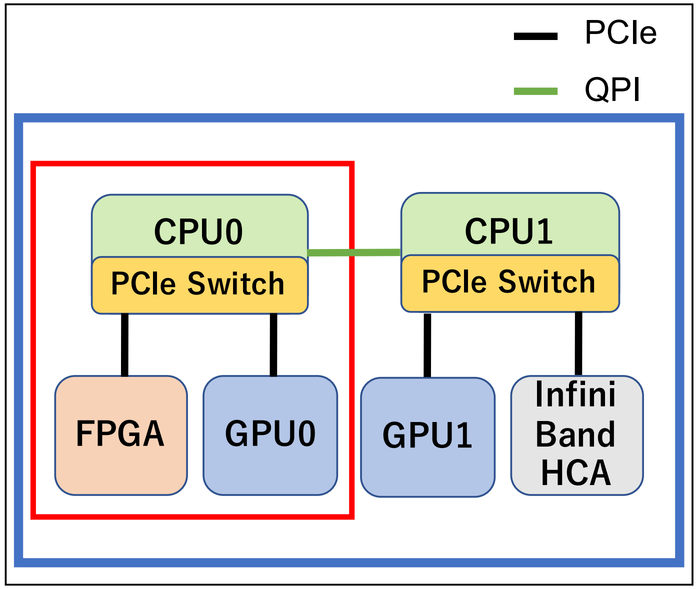
ARGOT method in GPU : OpenACC
ART method in FPGA: OpenACC

MHOAT

Evaluation Environment

- **PPX (Pre PACS-X) mini cluster which is the prototype of Cygnus**

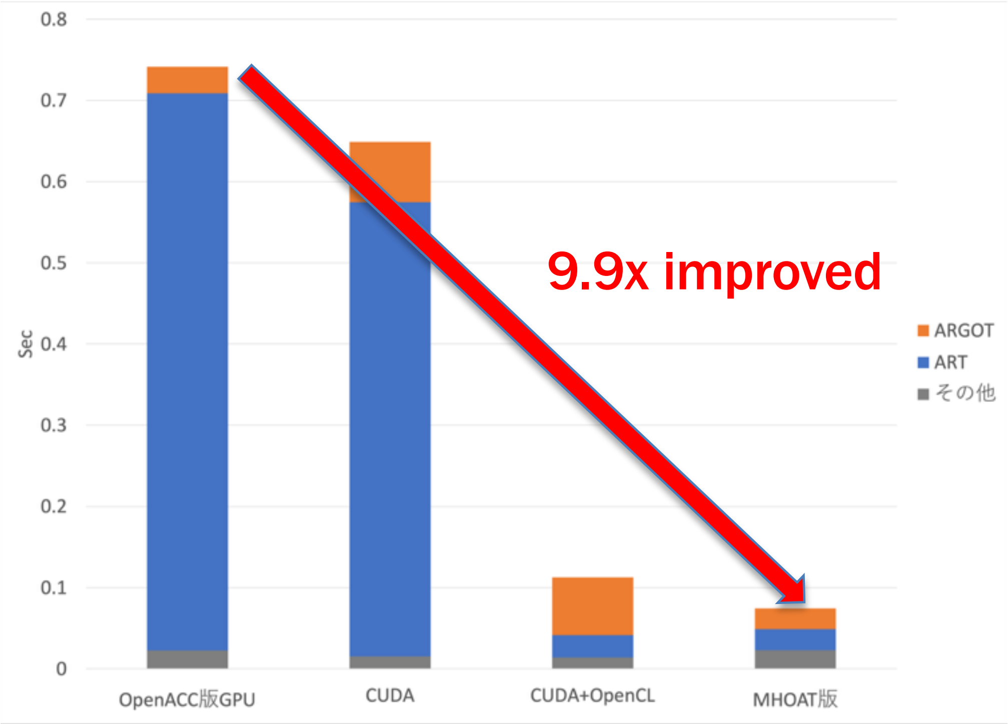
Hardware specification	
CPU	Intel Xeon E5-2690 v4 x2
Host Memory	DDR4-2400 8GB x8
GPU	NVIDIA Tesla P100 (PCIe Gen3 x16)
GPU Memory	16 GiB CoWoS HBM2 @ 720 GB/s with ECC
FPGA	BittWare 520N (1SG280HN2F43E2VG) equipped with 100Gbps x 4 chan.
FPGA Memory	DDR4 2400MHz 32 GB (8GB × 4)
InfiniBand	Mellanox ConnectX-4 Singleport EDR MCX455A-ECAT
Software specification	
OS	CentOS 7.9
Host Compiler	gcc 4.8.5
GPU Compiler	CUDA 11.2.152
Open MPI	3.1.0
OpenCL SDK	Intel FPGA SDK for OpenCL 19.4.0 Build 64 Pro Edition



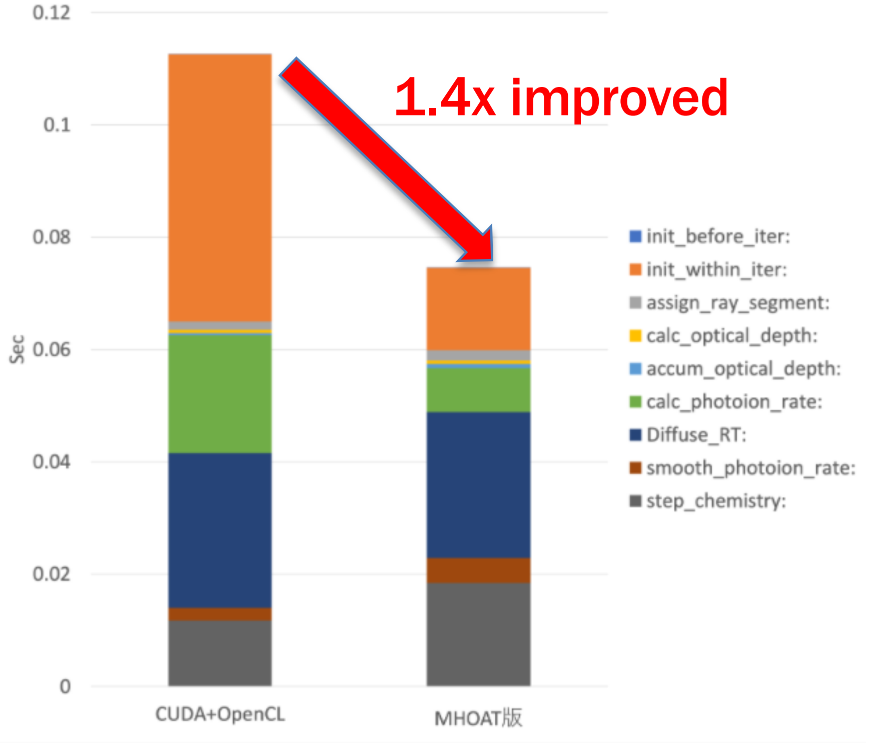
A computation node of PPX

PCIe for FPGA is just gen3 x 8 lanes, but does not affect on performance because the communication overhead is ~1%

Performance ：MHOAT vs CUDA/OpenCL vs GPU-only
(PPX single node: GPU: V100 + FPGA: Stratix10 size=32x32x32)



GPU (OpenACC or CUDA) vs CHARM (CUDA+OpenCL or MHOAT)
⇨ MHOAT is the best with 10x speed of OpenACC GPU



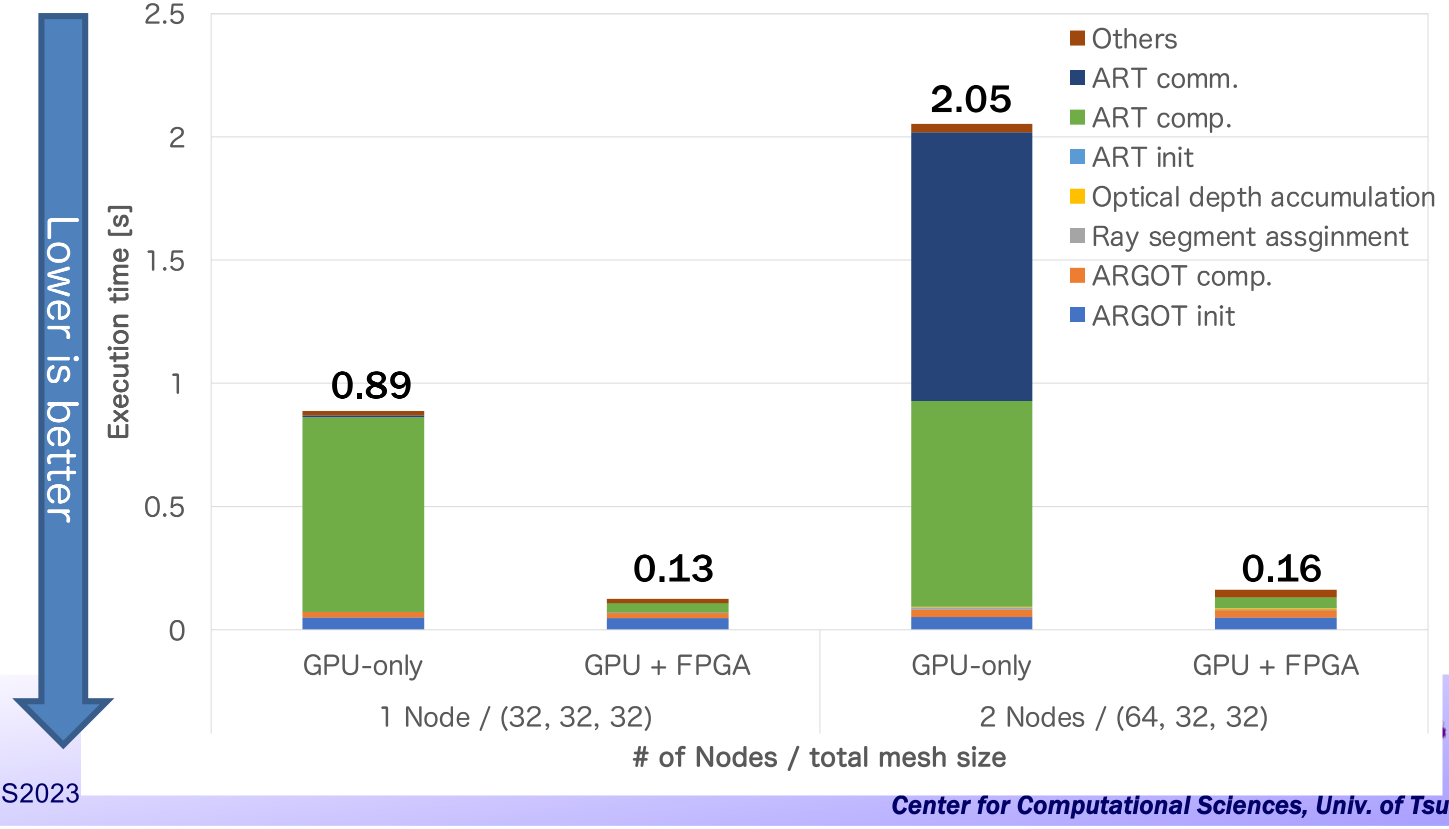
CHARM (CUDA+OpenCL or MHOAT)
⇨ equal with OpenCL on ART method, but ARGOT method is better with OpenACC than CUDA

Parallel ARGOT/CHARM with GPU+FPGA in oneAPI (CUDA+OpenCL)

- Weak scaling with 2 nodes (2 GPUs + 2 FPGAs) (preliminary evaluation)
  - largest mesh size on single FPGA is limited to 32^3
    - 1 node = 32x32x32 2 nodes = 64x32x32

- 1 node
  - GPU+FPGA achieves 6.8x faster than GPU-only
- 2 nodes
  - GPU+FPGA achieves 12.8x faster than GPU-only

  → combining computation and communication in pipeline



Summary

- Toward Exa-scale era, homogeneous or single accelerator system will have limitation on application variation and scalability
- CCS, U. Tsukuba, is running a multi-hetero supercomputer named Cygnus under CHARM (Cooperative Heterogeneous Acceleration with Reconfigurable Multi-devices) concept by GPU+FPGA
- Several supporting systems on FPGA and GPU coworking are developed including language solution toward high sustained performance of multi-physical simulations
- 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
- MHOAT makes complicated programming much easier
- Next Gen. HPC requires any of advanced technology, and FPGA will play an important role not just for the main engine for computation but also for compensation to many parts
- Fully parallelized ARGOT/CHARM is coming soon for full size Cygnus
- Call me if you want to use Cygnus with us!

My Research Team

- HPC Research Division at CCS, Univ. of Tsukuba
- Architecture Team in High Performance Computing System Lab., Dept. of Computer Science, Univ. of Tsukuba

Ryohei Kobayashi

Norihisa Fujita

Yoshiki Yamaguchi

Ryuta Tsunashima

- Astrophysics Research Division at CCS, Univ. of Tsukuba

Masayuki Umemura

Kohji Yoshikawa

THANK YOU !!

Q?