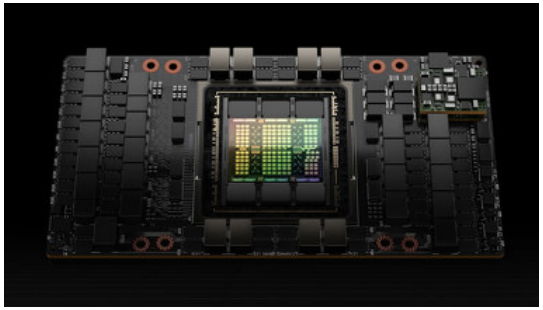
oneAPIによるGPU+FPGAマルチヘテロ環境プログラミングとアプリケーション実行

朴　泰祐
筑波大学・計算科学研究センター

（本発表の共同研究者：柏野隆太，小林諒平，藤田典久）

HPCにおけるアクセラレータ：GPUが席巻

- GPUだけでいいか？
  - ベクトル計算型の数多くのアプリケーションで有効
  - 定型ループ，幅広いSIMD実行で高効率
    - SIMD (STMD)アーキテクチャによる大容量並列処理
    - 高バンド幅のHBM (HBM2, 2e, 3）と小容量だが高速なローカルメモリによるHPC計算
  - GPUだけの処理による限界
    - （局所的に）並列性が不十分な場合
    - 処理が定型的でない場合（条件分岐）
    - 粒度が小さくノード間通信が頻発する場合（CPU制御，kernel switch）



NVIDIA Tesla H100
Tensor Core GPU
(from NVIDIA web page)

FPGAのHPC向け利用

- 近年のハイエンドFPGA
  - 真の意味でアプリケーションとハードの codesigning
  - プログラミング環境： OpenCL, C++, etc.
  - 高性能の光インターコネクト: 100Gb x N
  - 柔軟な精度コントロールが可能
  - （比較的）低電力
- 課題
  - プログラマビリティ: OpenCLは不十分，性能不足
  - GPUより低い標準FLOPS
  - メモリテクノロジ，バンド幅: GPUより1世代程度遅い

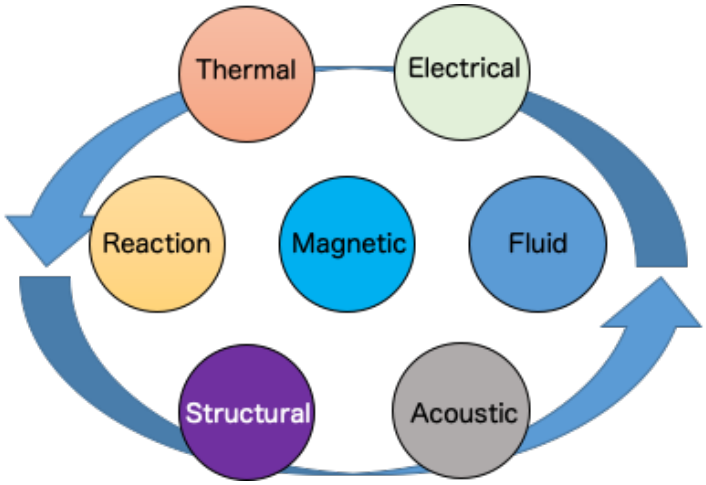
BittWare 520N with Intel Stratix10 FPGA equipped with 4x 100Gbps optical interconnection interfaces

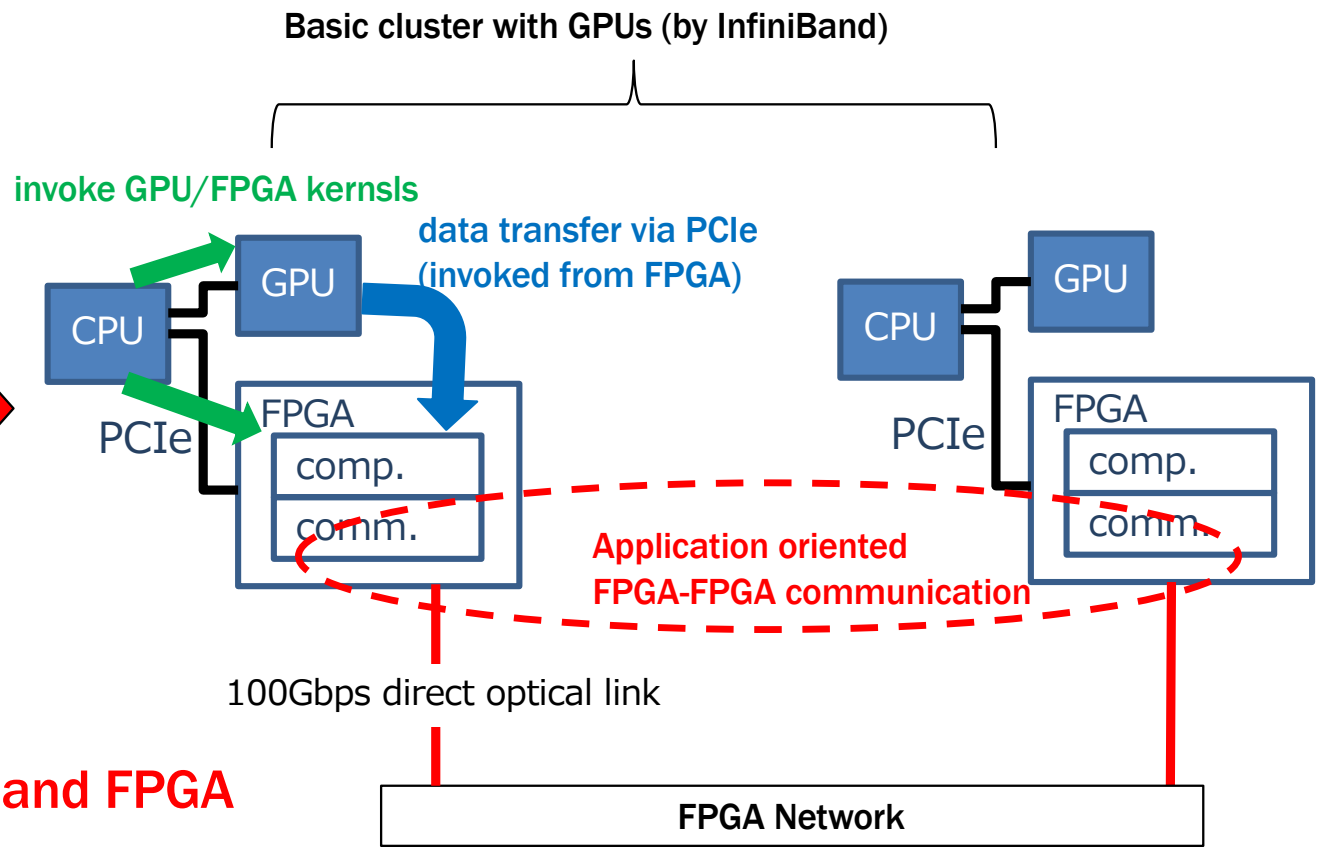
HPCソリューションとしてのGPUとFPGAの対照的性質

device	GPU	FPGA
parallelization	SIMD (x multi-group)	pipeline (x multi-group)
standard FLOPS	😀😀 (1000x cores)	😀 (~100x 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)	😀 (autonomic)
HPC applications	😀 (various fields)	😢 (not much)

CHARM: Cooperative Heterogeneous Acceleration with Reconfigurable Multi-devices

multi-physics/multi-scale complicated problem

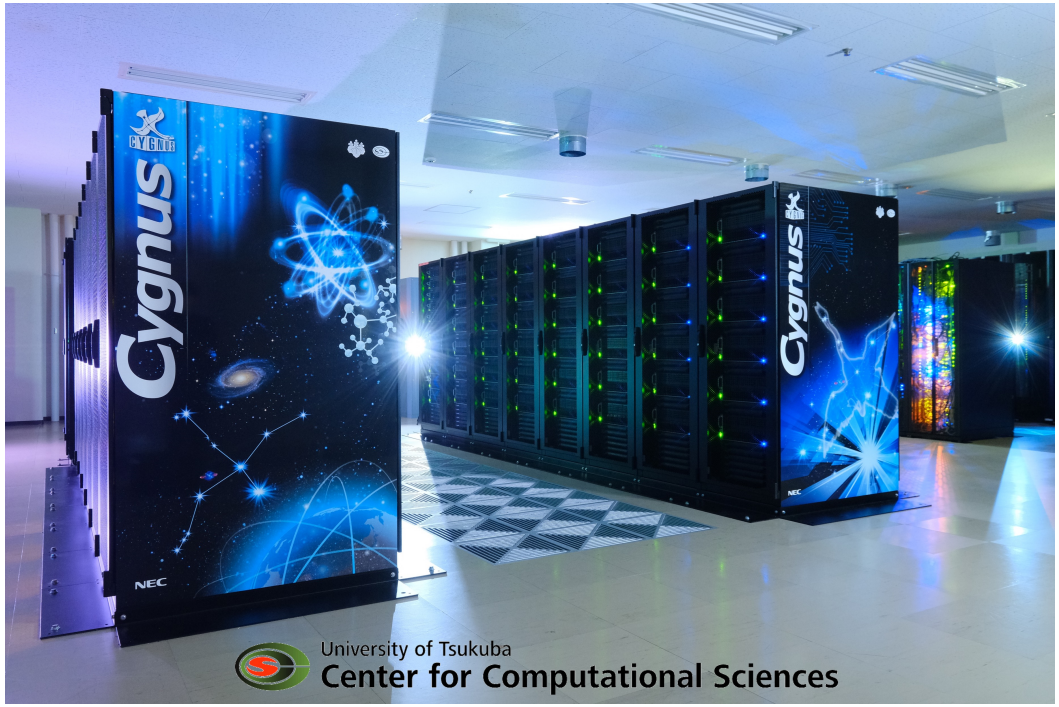




Cooperative computing with GPU and FPGA

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

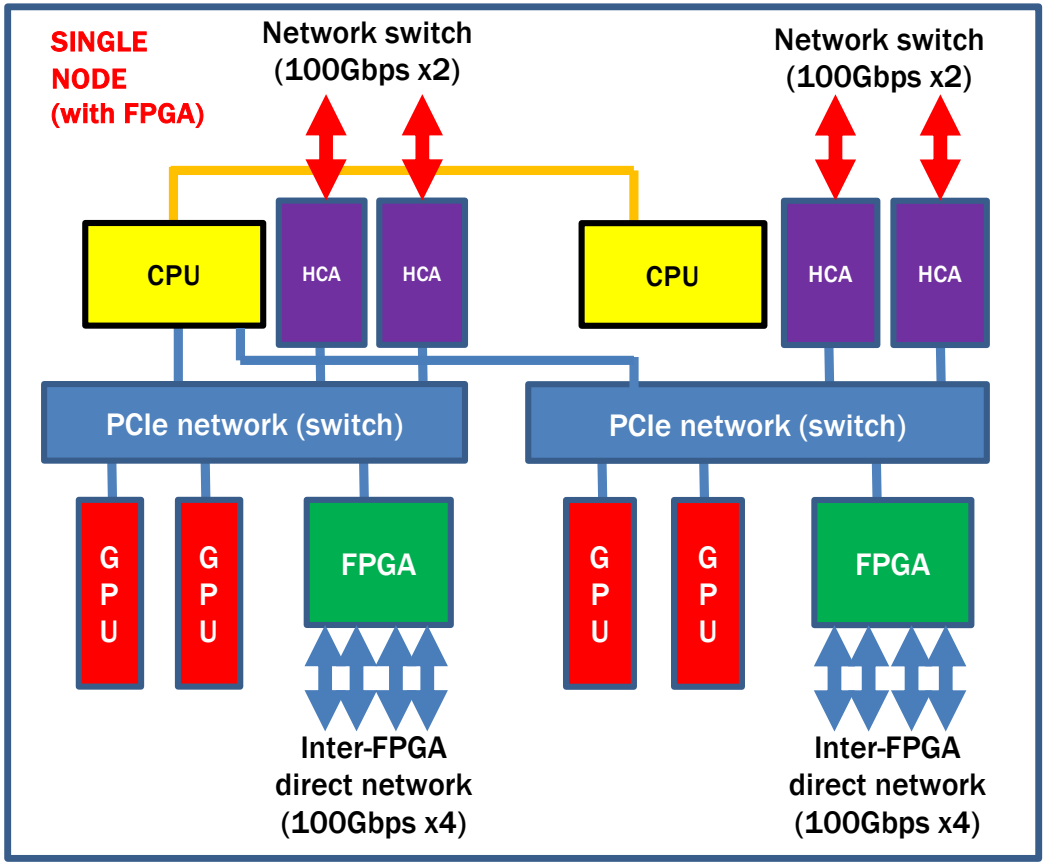
@ CCS, Univ. of Tsukuba (deployed by NEC)



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)

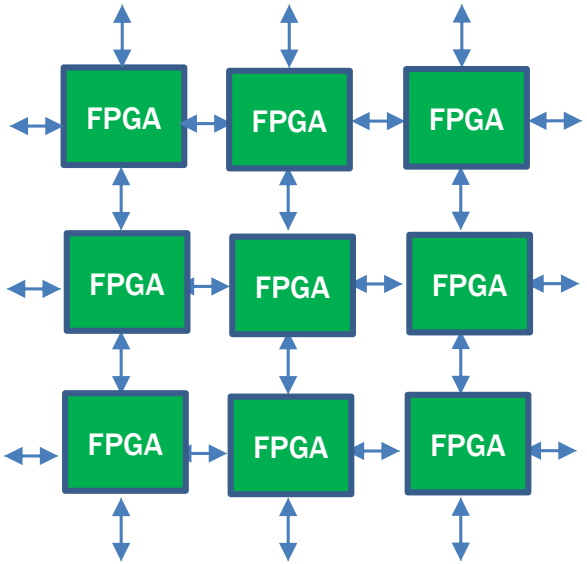
Single node configuration (Albireo)

- 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



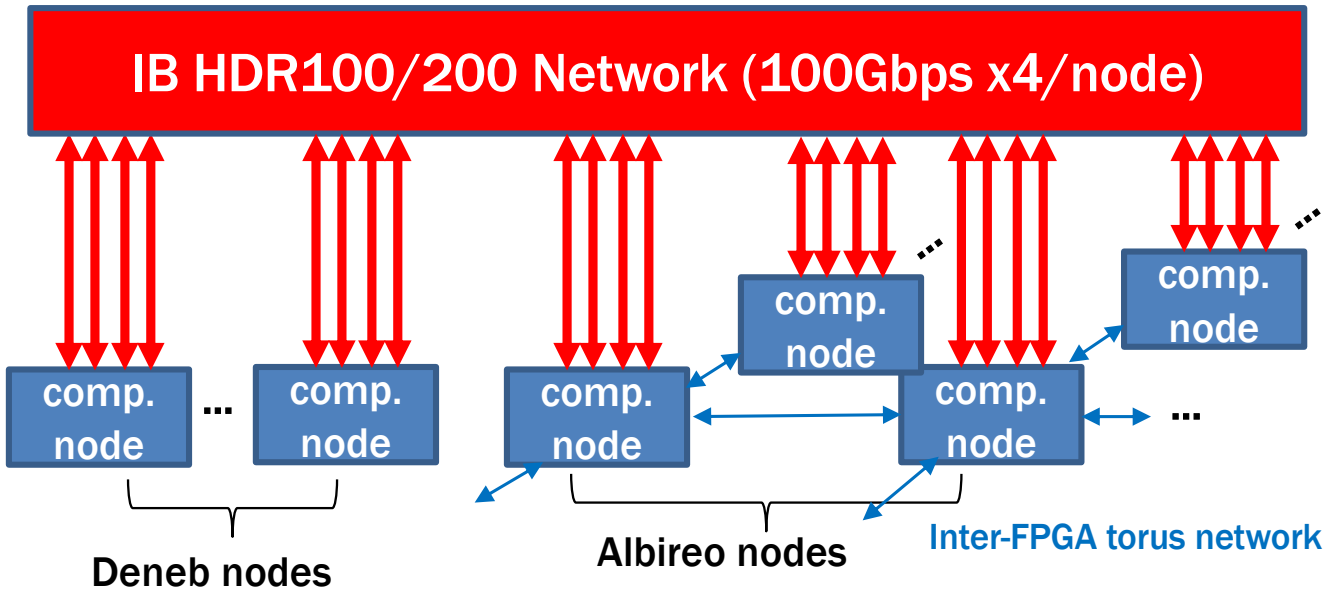
Two types of interconnection network

Inter-FPGA direct network (only for Albireo nodes)

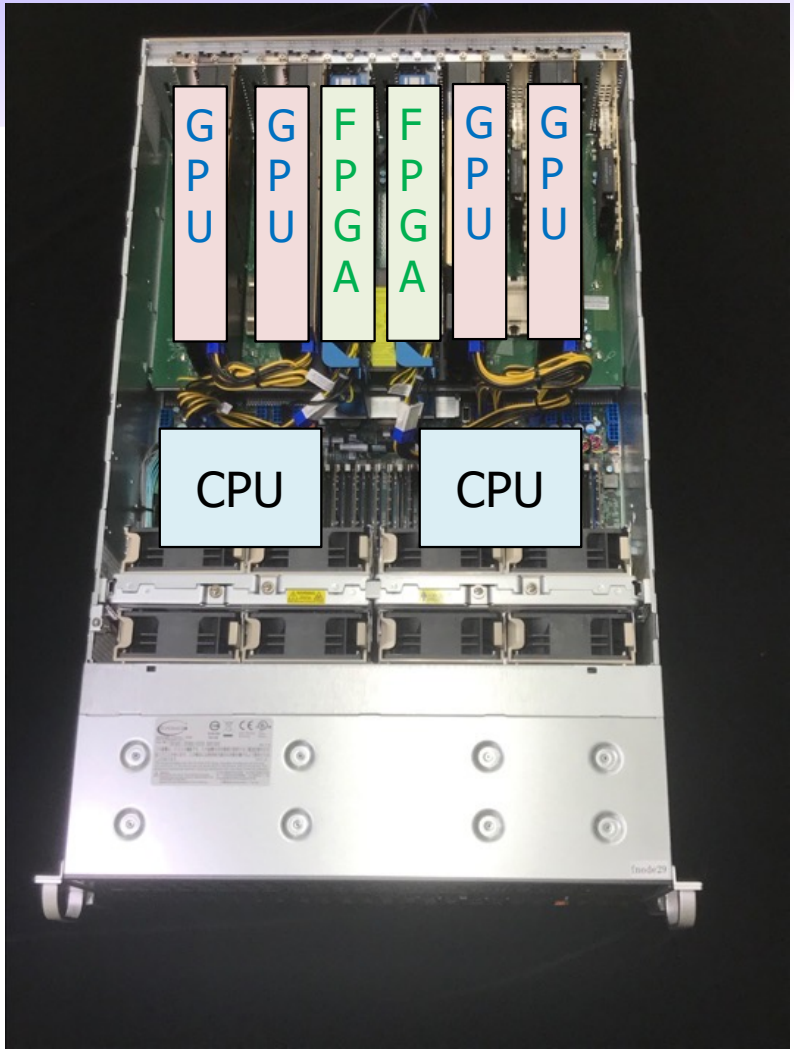


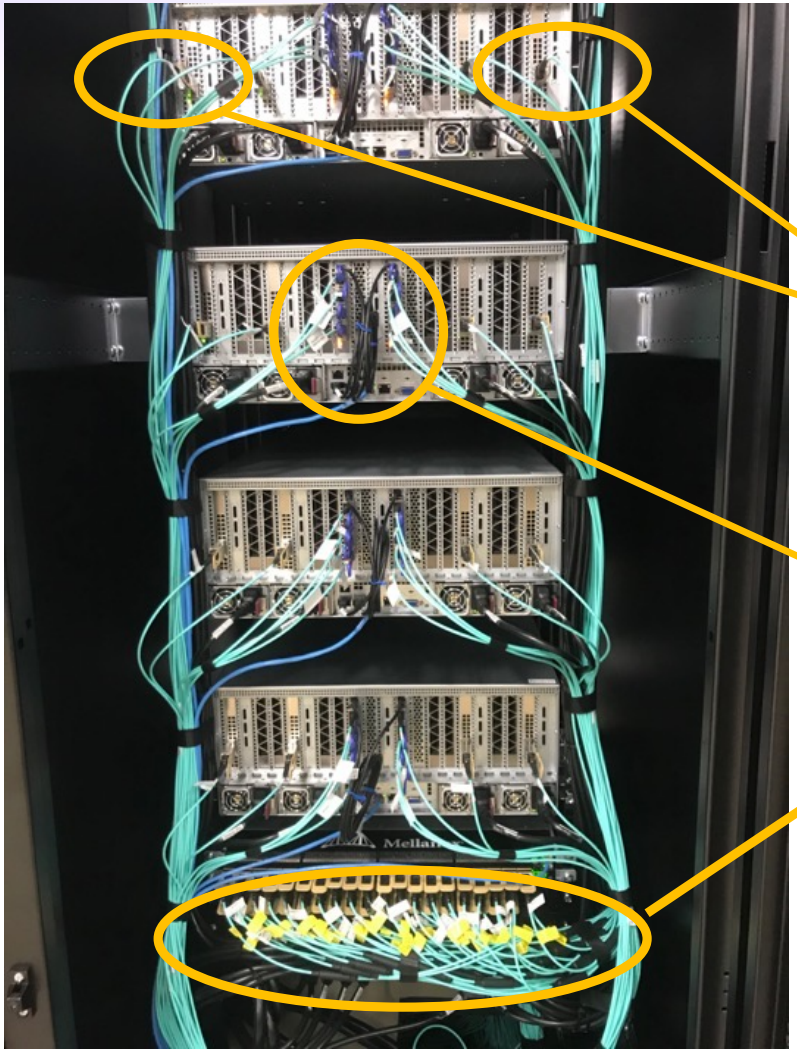
64 of FPGAs on Albireo nodes (2FPGAS/node) are connected by 8x8 2D torus network without switch

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

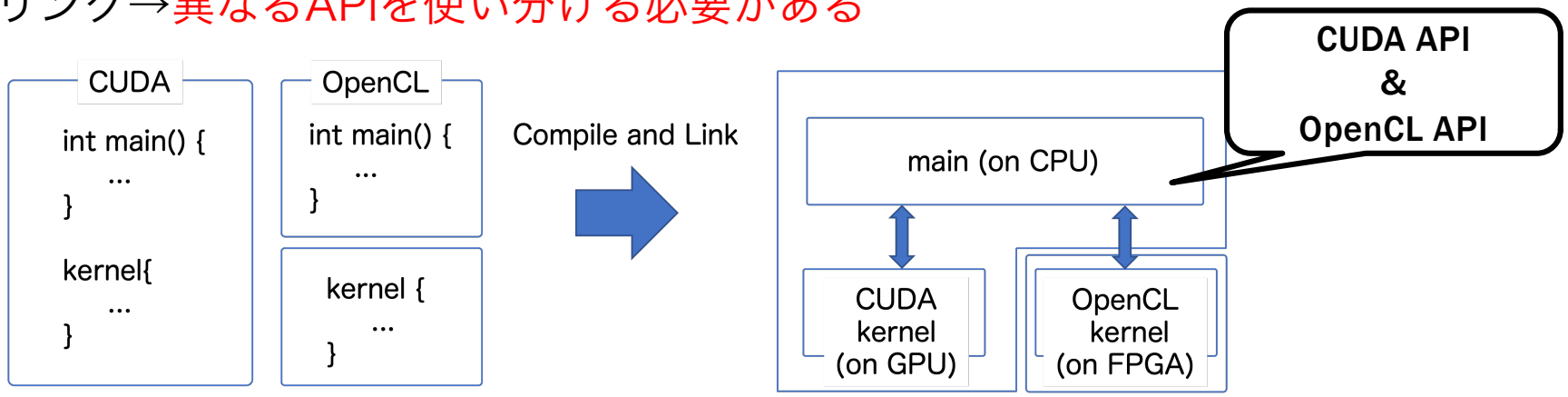
CHARMプログラミングの2つのアプローチ

- OpenACCのみによる記述
  - NVIDIA GPU compiler (PGI/NVIDIA) + FPGA research compiler (OpenARC@ORNL) を使う
  - OpenACC single codeに対し、拡張directiveによってGPUオフロードとFPGAオフロード部をユーザが明記、コードを分離して各バックエンドコンパイラに投げる
  - コードを解析して分離するメタコンパイラ：MHOAT (Multi-Hetero OpenACC Translater) を筑波大・ORNL・理研R-CCSで共同開発 *1
    ⇒ 開発途上でコンパイルはできるが性能が不十分
- 従来のGPU用コード、FPGA用コードをそのまま複合的に利用
  - GPU: CUDA（またはOpenACC）with NVIDIA compiler
  - FPGA: OpenCL with Intel SDK
  - ホストコード、CUDAコード、OpenCLコードが混在し見通しと記述性が悪い
    ⇒ コード資産は再利用しつつ見通しの良いオフローディング記述がしたい
    ⇒ Intel oneAPIを利用すれば可能だがGPUとFPGAの両方を使った実アプリケーション例がない

*1: Ryuta Tsunashima, et.al., “OpenACC unified programming environment for GPU and FPGA multi-hybrid acceleration”, HLPP2020

GPU・FPGA混合プログラムをどのように記述するか

- これまでのプログラミングモデル
  → CUDA + OpenCL混合プログラミング[*1]
  - CUDA（GPU用）とOpenCL（FPGA用）で記述された2つのプログラムを分割コンパイル&リンク→異なるAPIを使い分ける必要がある

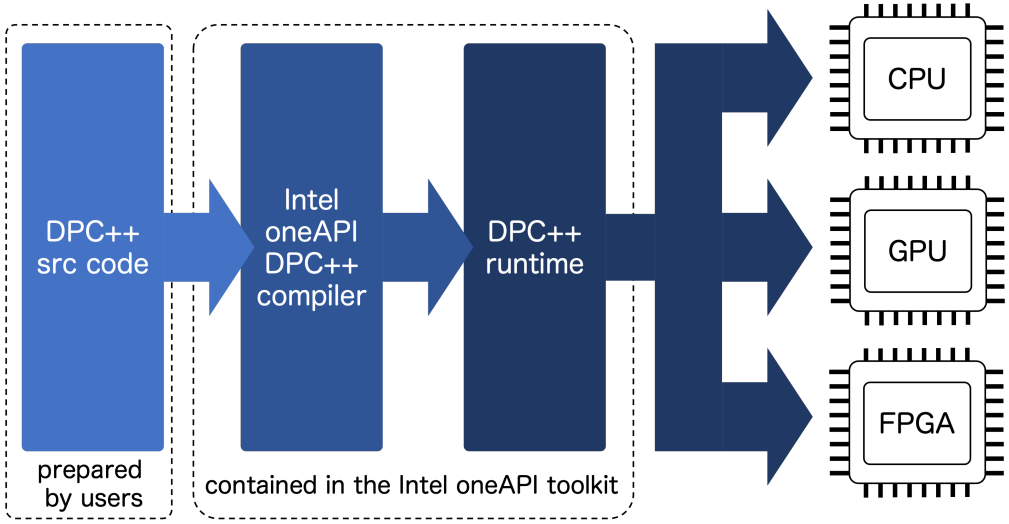


- コードの見通しを良くするためoneAPIを利用してCHARMを実装する
  - GPUとFPGAを統一的なスキームで制御できる

*1: Ryohei Kobayashi et al. “Accelerating Radiative Transfer Simulation with GPU-FPGA Cooperative Computation”, ASAP2020

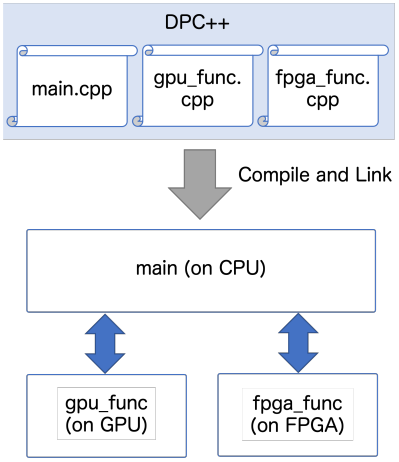
oneAPI概要

- oneAPIとは、インテルにより提唱された統合プログラミングモデル
  - 目的は異なるアーキテクチャ間における開発を簡略化・統一化すること
- oneAPIより、開発言語 Data Pallalell C++（DPC++）が提供されている
  - 異なるアーキテクチャ間で統一的な記述が可能

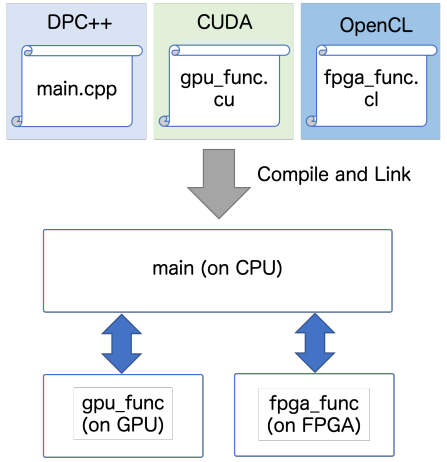


oneAPIによるCHARM実装

- DPC++でのプログラミングが推奨されている
  - (a) アプローチ
  - 問題：既存のGPUやFPGAのコードは、CUDAやOpenCLなど、他の言語で書かれている
  - DPC++による再実装はユーザーにとって負担
- oneAPIは他の言語で書かれたモジュールも使用可能
  - (b)アプローチ
  - コードの再利用が可能



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



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

oneAPI DPC++ Compiler

- oneAPI DPC++ コンパイラには2つのタイプがある.
  - Intel oneAPI Base Toolkit に付属する公式版
  - オープンソースとして公開されているOSS版[*1]

    NVIDIA サポートはOSS版のみ

    →本研究ではOSS版を利用する
- OSS版コンパイラ[*1]を利用して簡単なベクタ加算プログラムが，以下の環境で動作することを確認
  - NVIDIA GPU (V100), Intel FPGA (Stratix10)

*1: https://github.com/intel/llvm

oneAPIによるCHARM実装

CUDA + OpenCL を別々に記述・合成

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

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

oneAPIから CUDA+OpenCL カーネルを呼び出す

```
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(...);
```

oneAPIを使うことで、統一的APIによりGPUとFPGAの制御が可能

詳細：Ryuta Kashino, et.al., “Multi-hetero Acceleration by GPU and FPGA for Astrophysics Simulation on Intel oneAPI Environment”, HPC Asia 2022

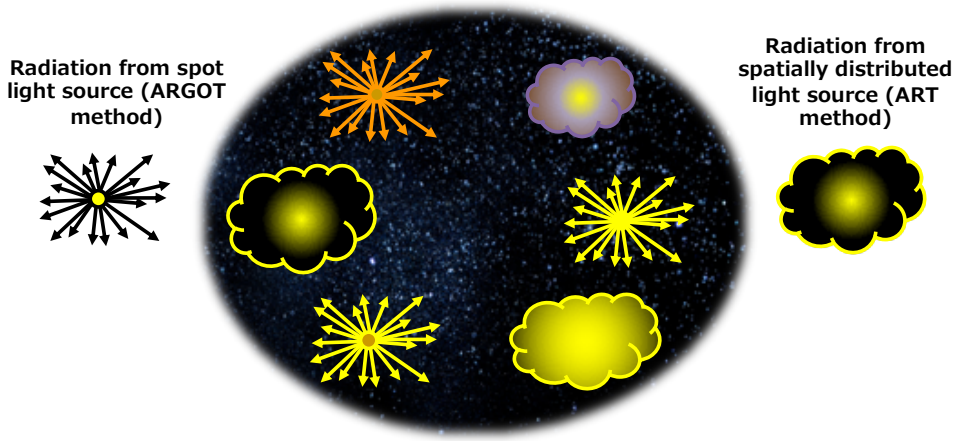
ターゲットコード: ARGOT

- ARGOT (Accelerated Radiative transfer on Grids using Oct-Tree)
  - 初期宇宙における天体形成をシミュレーション

    筑波大学計算科学研究センターにより開発
  - ２つの部分計算からなる

    ARGOT法
    - 点光源の輻射輸送
    - GPUが得意とする計算

    ART法
    - 空間に分散した光源の輻射輸送
    - GPUが不得意とする計算
    - FPGAによる高速化を行う

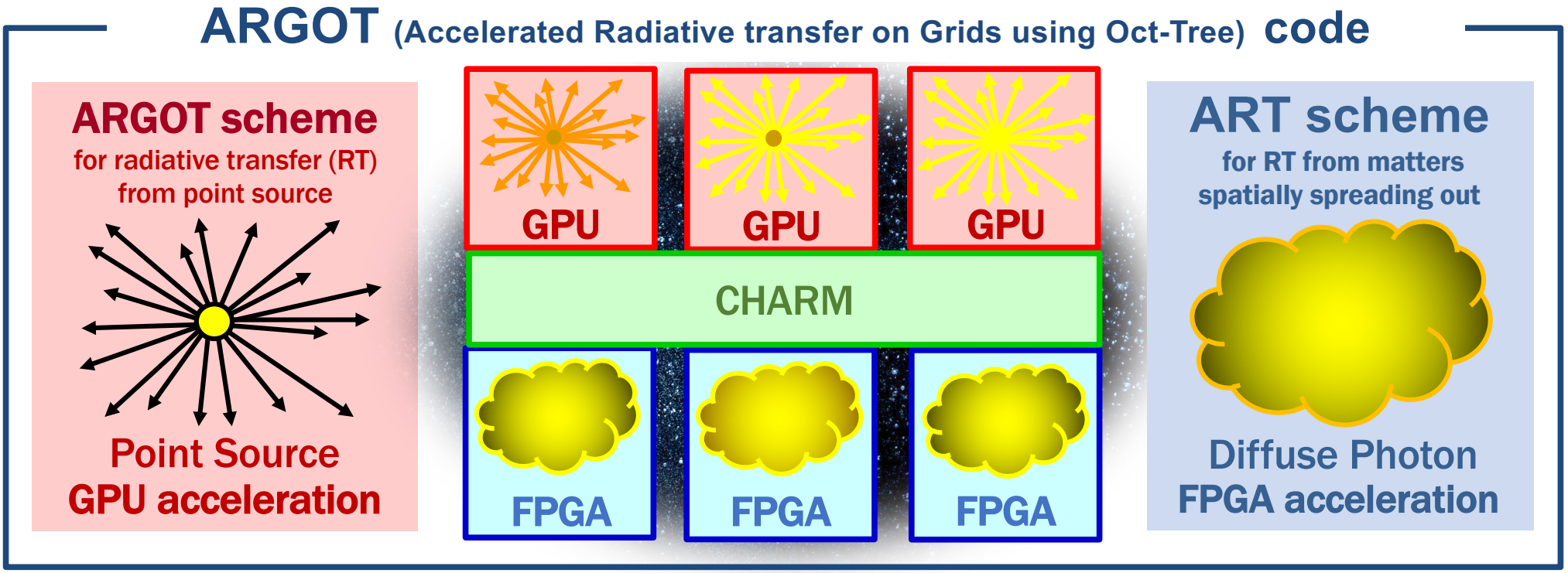


Two computation elements in ARGOT code: ARGOT method and ART method

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

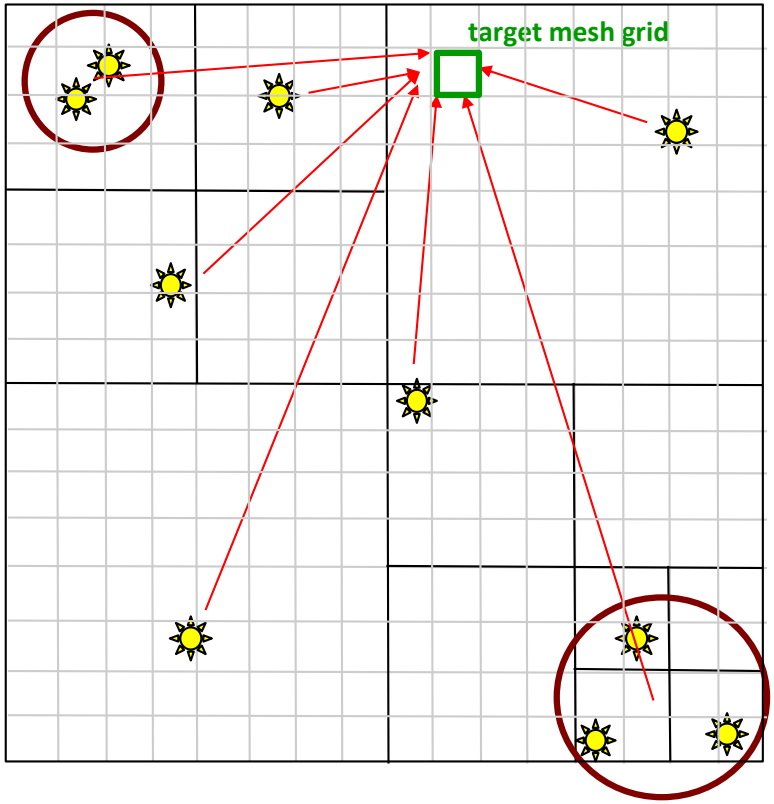
Two computation elements in ARGOT code: ARGOT method and ART method

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



ARGOT法　(CUDAによる実装)

- ARGOT法は点光源の輻射輸送を計算
- 八分木を用いて3次元空間に分散する点光源を表す
  - 遠距離にある点光源の集合を単一の光源とみなせる
    計算量: O(N^2) → O(NlogN)
- 重力計算における Tree-Code に似た手法
  - GPU実装に適している

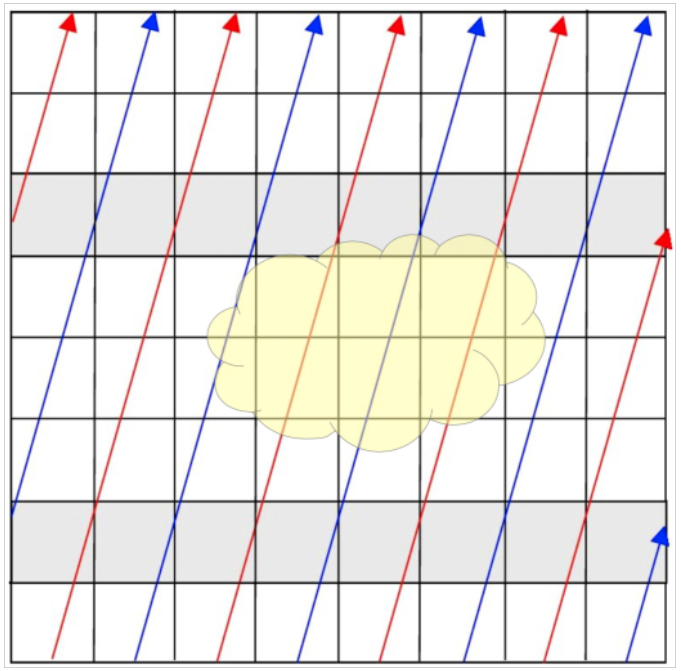


ART法(OpenCLによる実装)

- ART法は空間に広がる光源からの輻射輸送を計算
  - 問題空間の端からレイが並行に直進
- ART法はGPUに不向きな計算
  - レイの進行方向によってメッシュデータのメモリアクセスパターンが変化
    ランダムアクセスに近いアクセスパターン
  - Atomic演算が必要
  - 計算中に含まれる演算数が不十分

- FPGAを用いて高速化
  - FPGAのon-chipメモリを活用
  - 一定サイズのベクトル処理が独立に大量に必要
    ⇒ GPUの大規模SIMDが有効活用できない

ART法の実行時間は全体の90%を占める



詳細：**Norihisa Fujita, et.al., “Accelerating Space Radiative Transfer on FPGA using OpenCL”, HEART2018**

DPC++からCUDA/OpenCLカーネルをどのように呼び出すか

Example: CUDAカーネルの呼び出し

```
sycl::gpu_selector gpu_selector;
sycl::queue queue(gpu_selector);

dev_mem= malloc_device<>(..., sycl_queue);

queue.submit([&](handler& h) {
   h.memcpy(dev_mem, hmem, ...);
})

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

デバイスの選択

データ転送

カーネル実行

Example: OpenCLカーネルの呼び出し

```
clGetDeviceIDs(...,&dev,...)
ocl_ctx = clCreateContext(...,&dev,...);
prg = clCreateProgramWithBinary(ocl_ctx, ...);
ocl_kernel = clCreateKernel(prg,  ...);
ocl_queue = clCreateCommandQueue(ocl_ctx, dev,...);


sycl::context sycl_ctx(ocl_ctx);
sycl::kernel sycl_kernle(ocl_kernel, sycl_ctx);
sycl::queue queue(ocl_queue, sycl_ctx);

dev_mem= malloc_device<>(..., sycl_queue);
queue.submit([&](sycl::handler &h) {
      h.memcpy(dev_mem, hmem, ...);
});

queue.submit([&](sycl::handler &h) {
      // Call the OpenCL kernel
      h.set_args(dev_mem,...);
      h.single_task(sycl_kernel);
});
```

DPC++からCUDA/OpenCLカーネルをどのように呼び出すか

Example: CUDAカーネルの呼び出し

```
sycl::gpu_selector gpu_selector;
sycl::queue queue(gpu_selector);

dev_mem= malloc_device<>(..., sycl_queue);

queue.submit([&](handler& h) {
    h.memcpy(dev_mem, hmem, ...);
})

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

Example: OpenCLカーネルの呼び出し

```
clGetDeviceIDs(...,&dev,...)
ocl_ctx = clCreateContext(...,&dev,...);
prg = clCreateProgramWithBinary(ocl_ctx, ...);
ocl_kernel = clCreateKernel(prg,  ...);
ocl_queue = clCreateCommandQueue(ocl_ctx, dev,...);
```

OpenCLオブジェクトの生成

```
sycl::context sycl_ctx(ocl_ctx);
sycl::kernel sycl_kernle(ocl_kernel, sycl_ctx);
sycl::queue queue(ocl_queue, sycl_ctx);
```

DPC++オブジェクトへ変換

```
dev_mem= malloc_device<>(..., sycl_queue);
queue.submit([&](sycl::handler &h) {
      h.memcpy(dev_mem, hmem, ...);
});
```

データ転送

```
queue.submit([&](sycl::handler &h) {
      // Call the OpenCL kernel
      h.set_args(dev_mem,...);
      h.single_task(sycl_kernel);
});
```

カーネル実行

oneAPIによるARGOT法(CUDA kernel)

- ARGOT法はCUDAで実装
  - DPC++からCUDAを呼び出す
- “oneAPI for CUDA”を利用
  - Codeplayにより開発
    OSS版DPC++コンパイラでのみ利用可能
  - メモリ転送やカーネル実行は右図の通り

DPC++ code on CPU

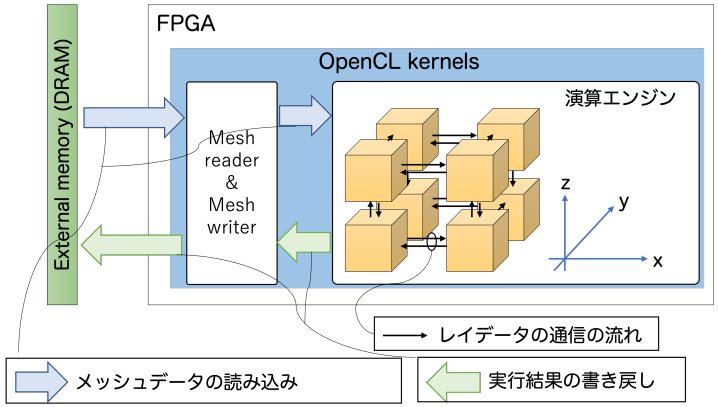
```
cuda_mem[idev].segment_dev = sycl::malloc_device<struct
ray_segment>(nseg_to_dev[idev], sycl_gpu->strm_queue[idev]);
...
sycl_gpu->strm_queue[idev].submit([&](sycl::handler& h) {
  h.memcpy(cuda_mem[idev].segment_dev, seg+offset_seg[idev],
sizeof(struct ray_segment)*nseg_to_dev[idev]);
});
sycl_gpu->strm_queue[idev].wait();
...
while(nseg_waiting > 0) {
  ...
  dim3 nthrd(nthread, 1, 1);
  dim3 nblck(nblock, 1, 1);

  sycl_gpu->strm_queue[idev].submit([&](sycl::handler& h) {
  h.interop_task([=](sycl::interop_handler ih) {
    // Call the CUDA kernel directly from SYCL
    calc_optical_depth_kernel<<<nblck, nthrd, 0>>>
      (cuda_mem[idev].mesh_dev, cuda_mem[idev].segment_dev,
                cuda_mem[idev].this_run_dev, offset);
  });
  });
  sycl_gpu->strm_queue[idev].wait();
  ...
}
```

memcpy (CPU -> GPU)

CUDA kernel launched

oneAPIによるART法(OpenCL kernel)

- ART法はOpenCLで実装
  - OpenCL実装の構成




```
// Construct SYCL version of the context
sycl::context sycl_context(ocl_context);

// Construct SYCL version of the kernel
sycl::kernel sycl_mesh_rw_kernel(ocl_mesh_rw_kernel, sycl_context);
sycl::kernel sycl_pe_kernel_0(ocl_pe_kernels[0], sycl_context);
sycl::kernel sycl_pe_kernel_1(ocl_pe_kernels[1], sycl_context);
...
sycl::kernel sycl_pe_kernel_7(ocl_pe_kernels[7], sycl_context);

// Construct SYCL version of the queue
sycl::queue sycl_mesh_rw_queue(ocl_mesh_rw_queue, sycl_context);
sycl::queue sycl_pe_queue_0(ocl_pe_queues[0], sycl_context);
sycl::queue sycl_pe_queue_1(ocl_pe_queues[1], sycl_context);
...
sycl::queue sycl_pe_queue_7(ocl_pe_queues[7], sycl_context);
```

- OpenCLオブジェクトをDPC++オブジェクトに変換している

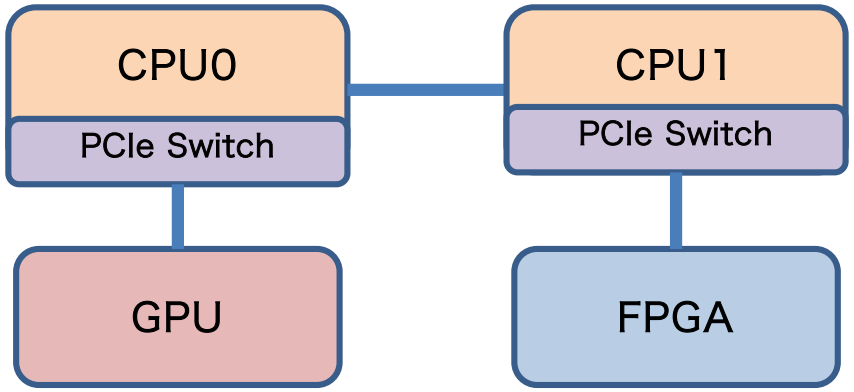
oneAPIによるART法(OpenCL kernel)

- 変換したDPC++オブジェクトに対してアクティベート操作を行う
  - Queueに対してカーネル実行を行っている
  - 同様にQueueに対してメモリ転送を行っている

```
  // OpenCL kernel launched
  /// mesh reader & writer
  sycl_mesh_rw_queue.submit([&](sycl::handler &h) {
                        h.set_args(d_rmesh);
                        h.single_task(sycl_mesh_reader_kernel);
                    });
  /// ray tracing (rt) processing elements
  /// rt_x0_y0_z0
  sycl_pe_queue_0.submit([&](sycl::handler &h) {
                        h.set_args(...);
                        h.single_task(sycl_pe_kernel_0);
                    });
...
  /// rt_x1_y1_z1
  sycl_pe_queue_7.submit([&](sycl::handler &h) {
                        h.set_args(...);
                        h.single_task(sycl_pe_kernel_7);
                    });


  // memcpy (FPGA -> CPU)
  sycl_mesh_rw_queue.submit([&](sycl::handler &h) {
                        h.memcpy(h_rmesh, d_rmesh, ….);
                       });
  sycl_mesh_rw_queue.wait();
}
```

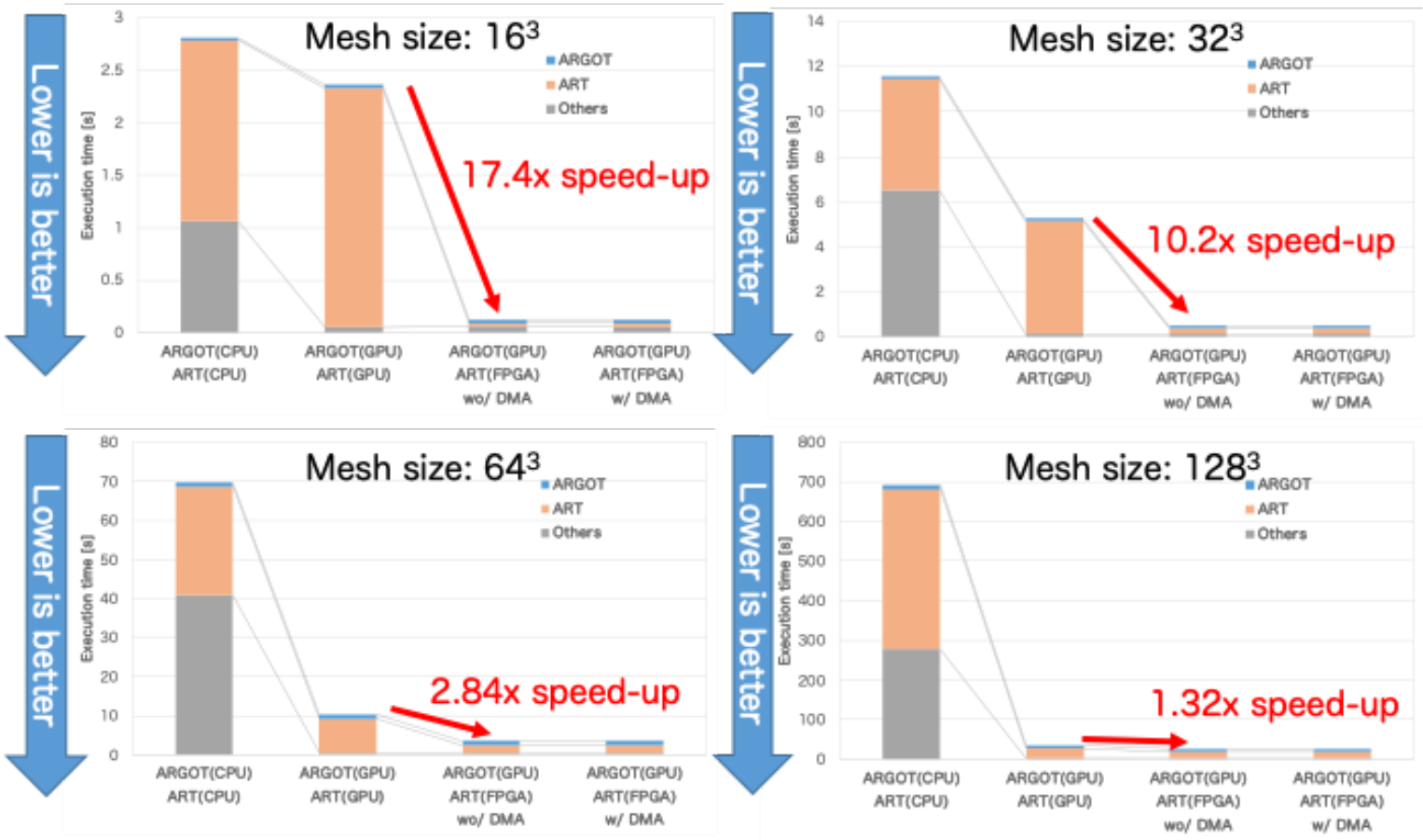
実験環境

- PPX (Pre PACS version X)
  - GPU・FPGA混載ノードをもつヘテロジニアス並列計算機
  - 筑波大学計算科学研究センターにより運用されている



Hardware specifications	
CPU	Intel® Xeon® Gold 6242 × 2
GPU	NVIDIA Tesla V100 (PCIe Gen3 x16 card version)
FPGA	Intel FPGA PAC D5005 (Intel Stratix 10 SX)
Software specifications	
Host OS	CentOS 7.9
Linux Kernel Version	3.10.0-1160.15.2.el7.x86_64
Compiler	oneAPI data parallel C++ compiler [7] (commitID: 6e9ddb6)
MPI	Open MPI 4.0.3
Accelerator Platforms	CUDA 10.2.89 Intel FPGA SDK for OpenCL 2021.2.0 Build 268.1 Pro edition

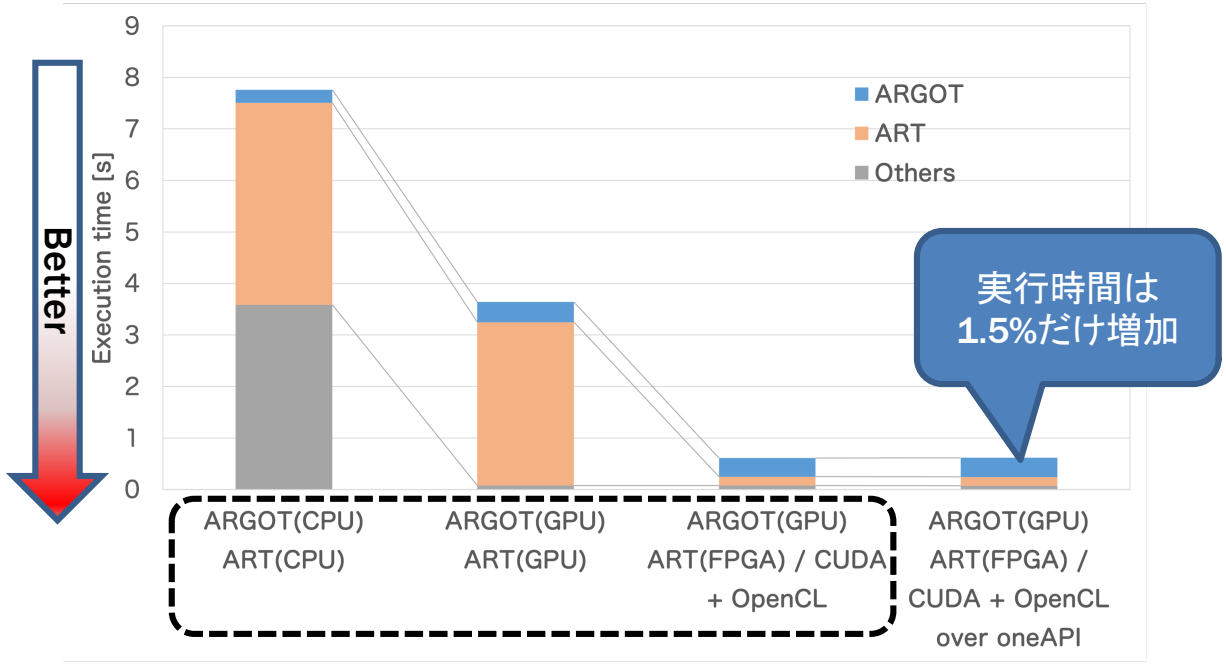
GPU-only vs GPU-FPGA 協調計算（ARGOT全体）



R. Kobayashi, et. al., “Accelerating Radiative Transfer Simulation with GPU-FPGA Cooperative Computation”, ASAP2020, Jul. 2020.

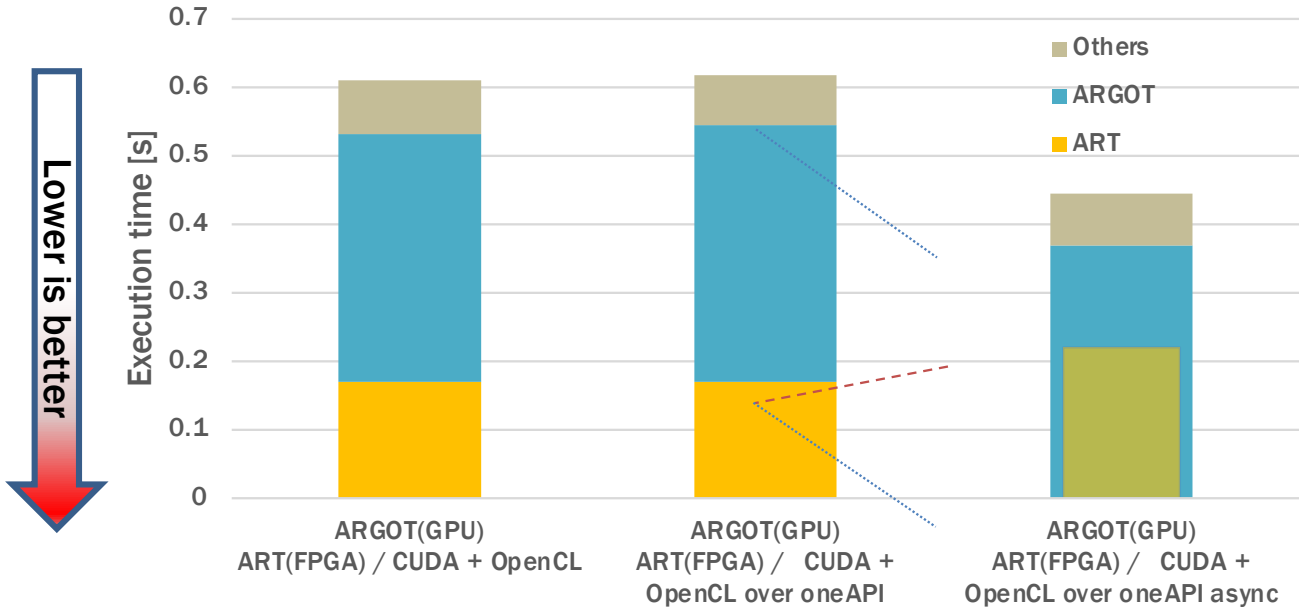
性能評価

- 問題サイズ: 32^3
- Others
  - ベクトル並列化が可能
    GPUによる高速化が効果的
- ART
  - ART法のGPU実装はCPUと同程度の性能
  - FPGAにより高速化
- GPU:CUDA+FPGA:OpenCL
  - GPUのみの場合と比べて10xの性能向上
- oneAPI vs CUDA+OpenCL
  - oneAPI版の実行時間が1.5%増加
    →oneAPI処理のオーバヘッドは無視できる



ARGOT法およびART法の非同期実行

- 本来ARGOT法とART法は2種類の物理現象を独立に計算し、最後に結果を足し合わせている
- 各Queueは非同期に実行可能
  - OpenMPを利用して，各Queueにactionを非同期に投入
  - Queueに投入されたactionは非同期実行される
- ART法の実行時間は ARGOT法の実行時間に隠蔽される
  - 全体として1.38xの高速化



*柏野隆太他, “oneAPIを用いたGPU・FPGA混載ノードにおける宇宙物理シミュレーションコードARGOTの実装”, 第183回HPC研究会

まとめ

- 筑波大学計算科学研究センターではGPU-FPGA協調計算である CHARM (Cooperative Heterogeneous Acceleration with Reconfigurable Multidevices)コンセプトを実現する様々な方法を研究している
  - プログラミングが最も大きい課題である
- oneAPIはGPU-FPGA協調計算を実現するためのプログラミングモデルの一つであると考えられる
- 既存のCUDA+OpenCLで記述されたARGOTコードをoneAPIに移植し，追加オーバーヘッドが無視できることを確認
  - また，Queueの非同期実行により性能向上が図れることを確認
  - この手法は，既存のHPCアプリケーションにCHARMのコンセプトを適応させる際に役立つ
- 将来的に世界初のヘテロジニアス（GPU＋FPGA）スーパーコンピュータCygnusでも積極的に活用する予定

  ⇒ CygnusでのoneAPI実行がいよいよ利用可能になった

  ⇒ OpenCLベースのFPGA間直接通信ツールCIRCUSとGPU側のMPIを組み合わせ、並列マルチヘテロ演算加速による実アプリケーション実行が可能に

研究チーム

- 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 Kashino


Ryuta Tsunashima

- Astrophysics Research Division at CCS, Univ. of Tsukuba


Masayuki Umemura


Kohji Yoshikawa