Enabling OpenACC Programming on Multi-hybrid Accelerated Cluster with GPU and FPGA

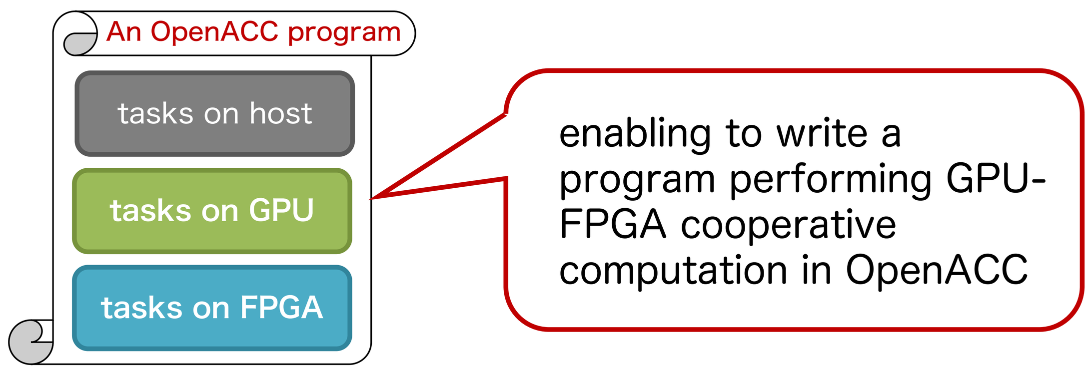
Ryuta Tsunashima, Ryohei Kobayashi, Nrihisa Fujita, Ayumi Nakamichi, Taisuke Boku (University of Tsukuba, Japan)
Seyong Lee, Jeffrey Vetter (Oak Ridge National Laboratory, USA)
Hitoshi Murai, Mitsuhisa Sato (RIKEN Center for Computational Science, Japan)

❖ Motivation

- GPU is the most popular Accelerator in HPC
  - large scale SIMD (SIMT) fabric, high bandwidth memory
  - But GPUs do not work well on application that employs
    - (partially) poor parallelism
    - non-regular computation (warp divergence)
    - frequent inter-node communication

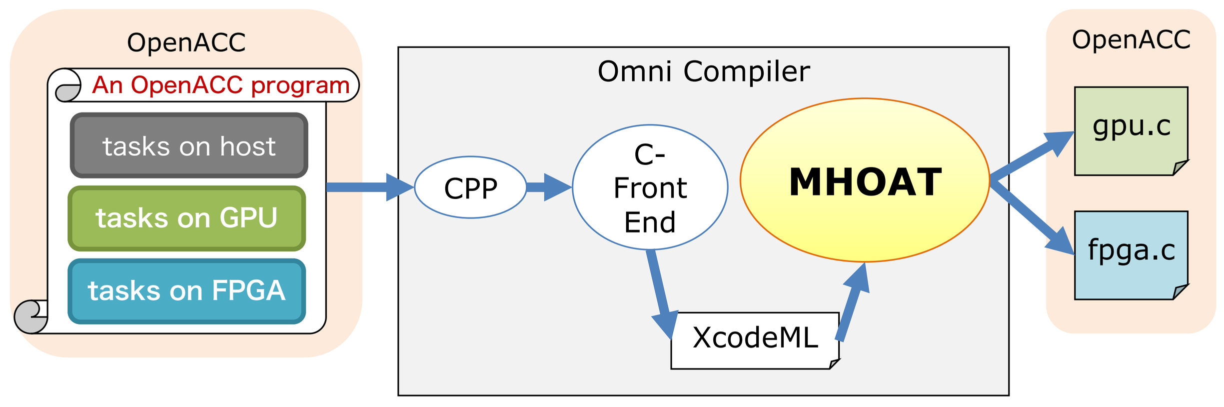
- FPGAs have been emerging in HPC
  - true co-designing with applications
  - making use of not only SIMD but also pipelining
    - effectively processing partially poor parallelism
  - high bandwidth interconnect: ~100 Gbps x 4

- We are challenging Multi-hybrid Accelerated Computing with GPU and FPGA
- However, currently users have to describe programs in **two languages** on different devices: ex) CUDA for GPU and OpenCL for FPGA
  - **causing heavy effort for users**
- We are building a uniform programming framework to make both devices work together at a single code by OpenACC
  - OpenACC is an API with directives for C, C++, Fortran for offloading to Accelerators
    - high level abstraction more than CUDA or OpenCL
  - **easier solution than describing programs in tow languages**



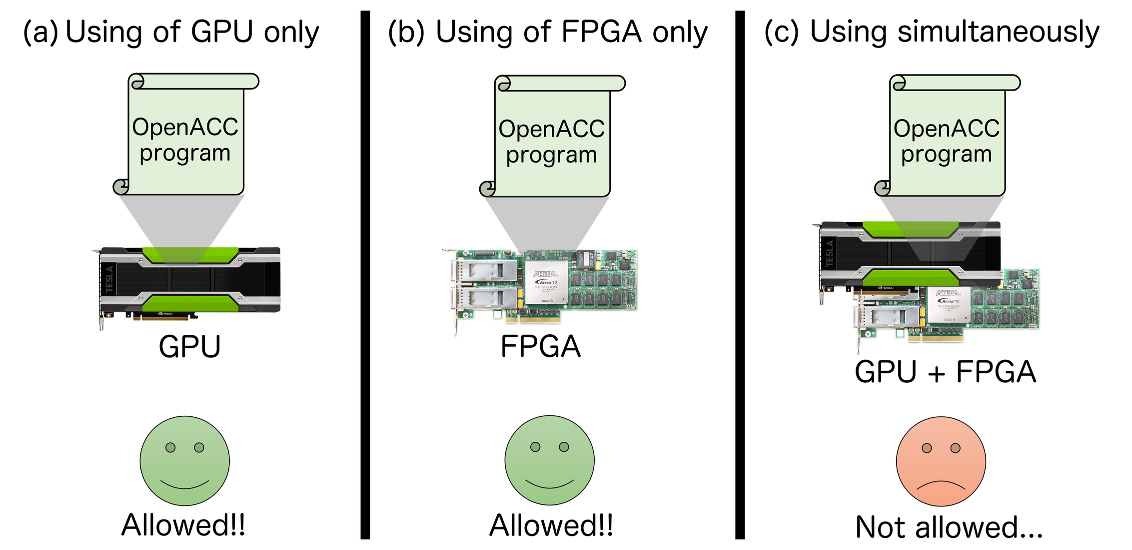
❖ MHOAT : meta-compiler

- MHOAT : Multi-Hybrid OpenACC Translator (Meta-compiler)
  - currently supporting C (because OpenARC allow input only C)
  - **under development with restricted functionality**
- Implemented with Omni Compiler developed by RIKEN R-CCS and CCS of University of Tsukuba
  1. Code is processed by CPP (C Preprocessor), then
  2. Translated to intermediate code called "XcodeML" by C-FrontEnd, and
  3. Compiled by MHOAT
- Input : A single OpenACC program with directive to specify target devices
  - **We extended current OpenACC directive with**
    - **#pragma accomn ondevice(*DEVICE*)**
      - "accomn" means extension in Omni Compiler
      - ***DEVICE*** is ***GPU*** or ***FPGA*** (predefined)
- Splitting the corresponding OpenACC-directed parts out of original code into two parts for GPU and FPGA

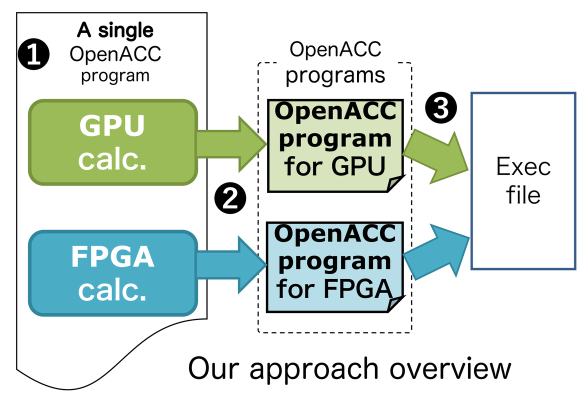


❖ Unified Programming Framework

- how to use both devices simultaneously
  - Current OpenACC compiler does not assume this situation



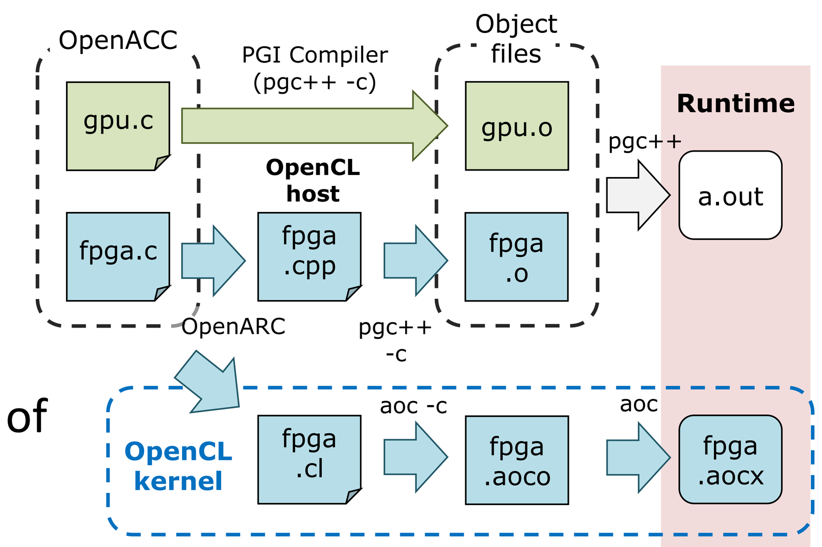
- Generating OpenACC programs for both devices and separately compiling them into an executive binary file



Our approach overview

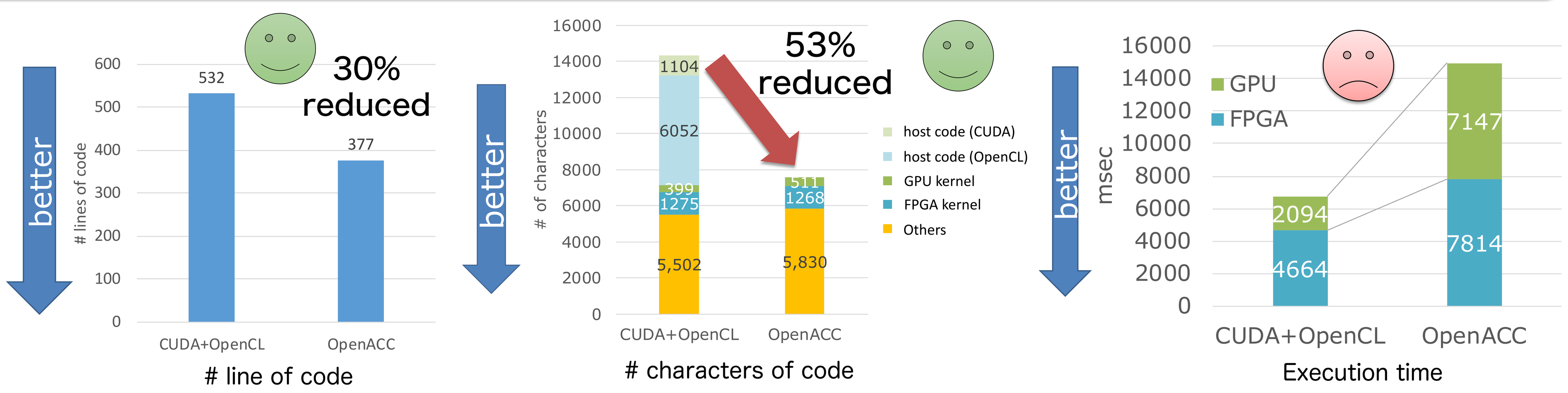
① Implementing a single OpenACC program
• Specifying target devices
② Generating OpenACC programs each for two devices
③ Final compilations are performed by appropriate backend compiler for each device

❖ Backend Compiler



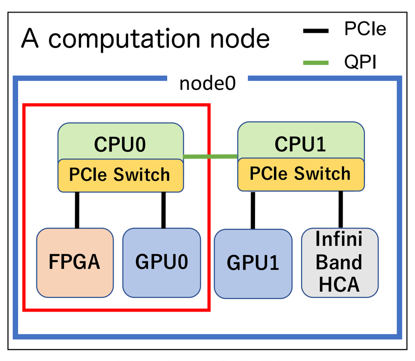
- Tow backend compilers
  - OpenARC **for FPGA**
    - OpenARC: Open Accelerator Research Compiler developed by FTG at ORNL
    - Enabling OpenACC for FPGA programming
    - Translating OpenACC code in **C to OpenCL with C++,** then OpenCL code is compiled by background compiler, Intel FPGA SDK for OpenCL
  - PGI compiler **for GPU**
    - Supporting OpenACC with C, C++ and Fortran
      - **C++ for linking with OpenCL host**
    - Compiling OpenACC to an object file directly
- Output parts by MHOAT are compiled by corresponding backend compilers
- Finally, two object files are linked to a single executable file by PGI compiler

❖ Evaluation



Pre-PACS version X (PPX)

Hardware specification	
CPU	Intel Xeon E5-2660 v4 x2
Host Memory	DDR4-2400 16GB x4
GPU	NVIDIA Tesla P100 x2 (PCIe Gen3 x16)
FPGA	Intel Arria 10 GX 1150 (BittWare A10PL4) (PCIe Gen3 x8)
Software specification	
OS	CentOS 7.3
GPU Compiler	PGI Compiler 18.10
FPGA Compiler	OpenARC V0.14 (April, 2019)
Host Compiler	GCC 4.8.5
OpenCL Compiler	Intel FPGA SDK for OpenCL 17.1.2.304



Node of PPX

* This comparison was done by hand-compilation before implementing MHOAT, preparing two OpenACC programs assuming operation of MHOAT.

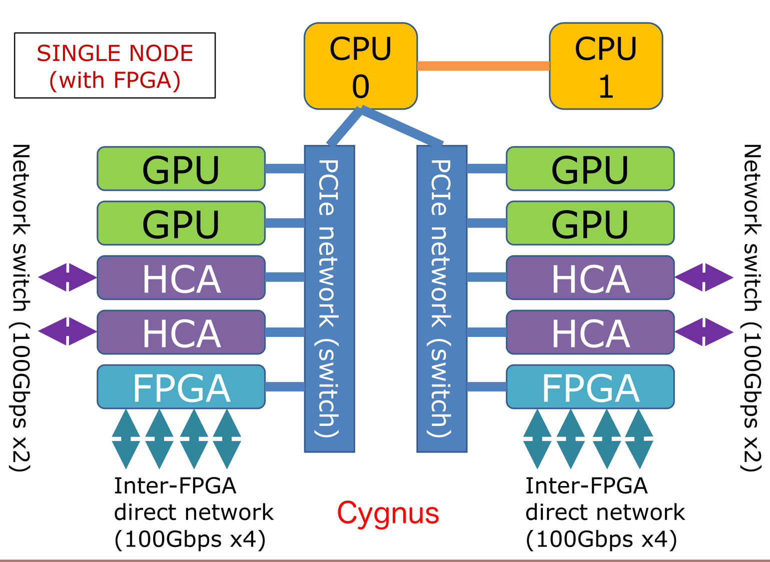
- Our approach vs. traditional one
  - Our approach: OpenACC*
  - Traditional approach: CUDA + OpenCL
- Using a toy program (NOT real application) **on PPX**
  - GPU : Performing matrix multiply
  - CPU : Receiving a GPU result and sending it to FPGA
  - FPGA : Performing the conjugate gradient method
- Programming cost comparison
  - # lines of code
    - Our approach **reduced 30% of LOC**
  - # characters of code
    - Our approach **reduced 53% of characters**
    - GPU kernel and FPGA kernel in OpenACC are corresponding to code blocks with directives
    - Others: init function, validation function, etc.
- Execution time comparison
  - GPU: 3.4x worse, FPGA: FPGA: 1.67x worse
    - Because of no performance tuning
    - Need to discuss FPGA parts with ORNL

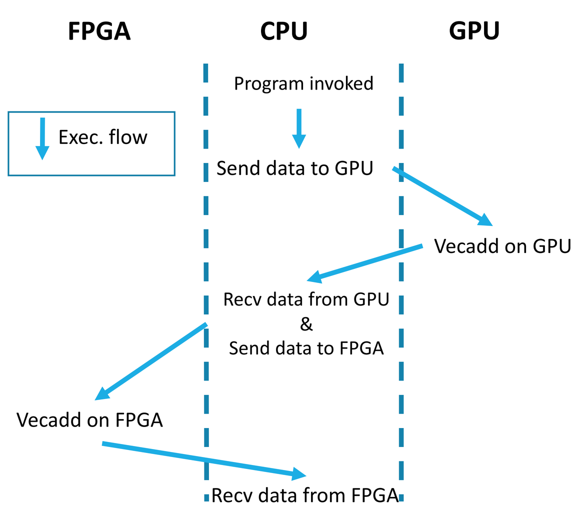
❖ Experiment of Compiling by MHOAT

- Realizing Multi-hybrid Accelerated Computing with GPU and FPGA from a process by an OpenACC program described under Unified Programing Framework
- Evaluation on a simplified synthetic code (NOT real application)
  - vector add on GPU, then result is used for vector add calculation on FPGA
  - Data communication between GPU and FPGA is perform via host memory (as shown)
- Result is verified with comparison on CPU version **on Cygnus at CCS**

Specification of Cygnus

Hardware specification	
CPU	Intel Xeon Gold 6126 (12C / 2.6GHz) x2
Host Memory	DDR4-2666 16GiB x12
GPU	NVIDIA Tesla V100 (32GiB HBM2 PCIe 3.0 x16) x4
FPGA	Intel Stratix 10 GX 2800 (BittWare 520N PCIe Gen3 x16) x2
Software specification	
OS	CentOS 7
GPU + Host Compiler	PGI Compiler 19.1
FPGA Compiler	OpenARC V0.17 (Oct, 2019)
OpenCL Compiler	Intel FPGA SDK for OpenCL 19.1.0.240





Program flow

ACKNOWLEDGEMENT This research is partially supported by "Communication-Computation Unified Supercomputing" project under MEXT's "Next Generation Supercomputer R&D" program and Collaborative Research between CCS, R-CCS and ORNL.