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

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 non-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 low languages.

MHOAT: meta-compiler
- MHOAT: Multi-Hybrid OpenACC Translator (Meta-compiler)
- Currently supporting C (Because OpenACC allow input only C).
- Under development with restricted functionality.
- Implemented with Omni Compiler developed by RIKEN R-CSS 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 acc on device DEVICE
    - "acc" 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.
    - Using GPU only
    - Using FPGA only
    - Using simultaneously

Evaluation
- Tow backend compilers
  - OpenARC for FPGA
    - OpenARC: Open Accelerator Research Compiler developed by FPG 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.
- Our approach vs. traditional one
  - Our approach: OpenACC
    - Traditional approach: CUDA + OpenCL.
  - Using a toy program (not real application) on PXP
    - 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.
  - Our approach
    - 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: 1.67x worse.
    - Because of no performance tuning.

Experiment of Compiling by MHOAT
- Realizing Multi-hybrid Accelerated Computing with GPU and FPGA from a process by an OpenACC program described under Unified Programming Framework.
- Evaluation on a simplified synthetic programs
  - For PXP version V0.17.
  - # of characters of compiled by PXP version Y.
  - Characters of code.
- Result is verified with comparison on CPU version on Cygnus at CCS.

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-CSS and ORNL.