CUDA vs OpenACC: Performance Case Studies

Tetsuya Hoshino, Naoya Maruyama, Satoshi Matsuoka

1. Background and Motivation

Heterogeneous Supercomputer
- Increase of heterogeneous system using many-core accelerators
  - e.g., Titan, Tsubame2.0, Stampede
- Porting of legacy CPU-based applications to accelerators has become a big challenge

**GPU Programming Interface**
- **CUDA**
  - The most widely used programming interface for GPGPU
  - Low-level programming is required
- **OpenACC**
  - A new accelerator programming interface
  - OpenMP-like loop directives
  - Support C and Fortran

**Motivation**
- To understand the performance of a new accelerator programming interface OpenACC

Evaluation Methodology
- Comparison of the performance of OpenACC and CUDA with case studies of porting and optimization

**Kernel Benchmarks**
- Matrix Multiplication
  - compute-bound kernel
- 7-point stencil
  - 3-D diffusion equation
  - memory-bound kernel

**A real-world application**
- A CFD application UPACS
  - developed by the Japan Aerospace Exploration Agency
  - nearly one hundred thousand lines of Fortran90
  - consists of three major computation phase
    - Convection (explicit)
    - Viscosity (explicit)
    - Time Integration (implicit)
  - Time Integration phase is parallelized by using hyperplane method

**Performance Evaluation**

**Evaluation Environment**
- Fermi GPU (M2050)
- 3 OpenACC compilers developed by PGI, Cray, and CAPS

**Experiments**
- Port and optimize each application
- Compare the performance and usability of OpenACC with CUDA

**Kernel Benchmarks**
- Matrix Multiplication
  - Matrix size: 2048^2
  - without counting the CPU-GPU transfer overhead
- 7-Point Stencil
  - 3-D array size: 256^3
  - without counting the CPU-GPU transfer overhead
  - PGI performance achieved 60%~98% of CUDA

**A real-world CFD application**
- Problem size: 120^3
- OpenACC performance
  - Baseline: achieved 64% of CUDA
  - Fully-optimized: achieved 42% of CUDA
  - Fine Grained Parallelization using shared memory is the most efficient optimization
  - But using shared memory is limited in OpenACC, it causes the performance gap

**Future Work**
- Improving compiler based optimizations by auto-tuning in the context of OpenACC without losing program portability