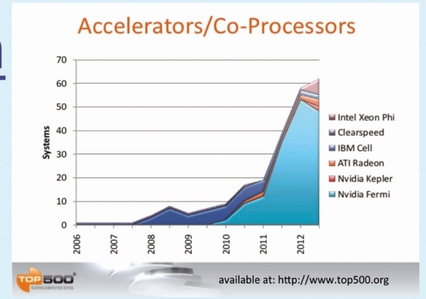
CUDA vs OpenACC : Performance Case Studies

Tetsuya Hoshino†, Naoya Maruyama†, ††, Satoshi Matsuoka†
† Tokyo Institute of Technology
†† RIKEN Advanced Institute for Computational Science

Background and Motivation

Heterogeneous Supercomputer

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



GPU Programming Interface

- CUDA・OpenCL
  - 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

TITAN (2012) 27.1 PFlops
299,088 CPU cores
18688 NVIDIA K20 GPUs

TSUBAME2.0 (2010) 2.28 PFlops
17,984 CPU cores
4258 NVIDIA Fermi GPUs

→ OpenACC matrix multiplication (baseline version)

```
!$acc kernels present(a, b, c)
!$acc loop
  do j = 1, n
!$acc loop
    do i = 1, n
      cc = 0
      do k = 1, n
        cc = cc + a(i,k) * b(k,j)
      end do
      c(i,j) = cc
    end do
  end do
!$acc end kernels
```

Motivation

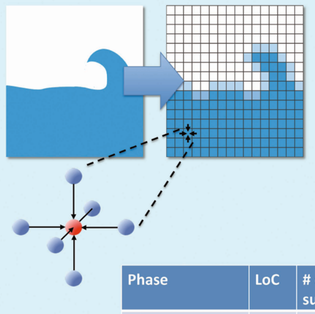
- To understand the performance of a new accelerator programming interface OpenACC

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



```
!$acc kernels present(f1, f2)
!$acc loop
  do z = 1, nz
!$acc loop gang vector(THREAD_Y)
    do y = 1, ny
!$acc loop gang vector(THREAD_X)
      do x = 1, nx
        w = -1; e = 1; n = -1; s = 1; b = -1; t = 1;
        if(x==1)  w=0; if(x==nx) e=0;
        if(y==1)  n=0;  if(y==ny) s=0;
        if(z==1)  b=0;  if(z==nz)  t=0;
        f2(x,y,z) = cc*f1(x,y,z) &
          + cw*f1(x+w,y,z) + ce*f1(x+e,y,z) &
          + cs*f1(x,y+s,z) + cn*f1(x,y+n,z)  &
          + cb*f1(x,y,z+b) + ct*f1(x,y,z+t)
      end do
    end do
  end do
!$acc end kernels
```

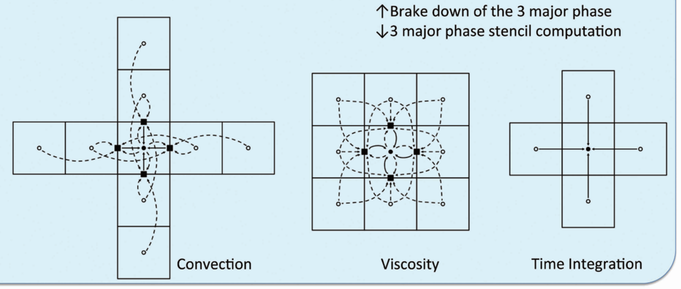
OpenACC 7-point stencil (Thread mapping PGI version)

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

Phase	LoC	# of subroutines	# of loop nests	# of 3-D data Read	Upload	Temporary	% of Total time (1CPU core)
Convection	682	10	7	29	5	15	25.0%
Viscosity	599	5	5	33	5	25	37.7%
Time Integration	622	5	5	20	5	5	28.5%

↑Brake down of the 3 major phase
↓3 major phase stencil computation



Performance Evaluation

Evaluation Environment

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

Machine (TSUBAME2.0 single node)

	CPU	GPU
Type	Intel Xeon Westmere-EP 2.9GHz	NVIDIA Fermi M2050
Cores	6 core	14SM, 448 CUDA core

Compiler

	compiler version	option
PGI	pgfortran 12.10-0	OpenACC: -O3 -acc -ta=nvidia,cc20,keepgpu CUDA Fortran: -O3 –Mcuda=cc20
Cray	ftn 8.1.0.165	-O3 -h accel=nvidia_20 -h system_alloc -h acc_model=fast_addr
CAPS	hmpp 3.2.4 host compiler ifort 11.1	--nvcc-options -arch,sm_20 --nvcc-options -O3 ifort -O3
CUDA C	nvcc 4.1	-O3 –arch=sm_20
Fortran	ifort 11.1	-O3 -static -xP -openmp

Experiments

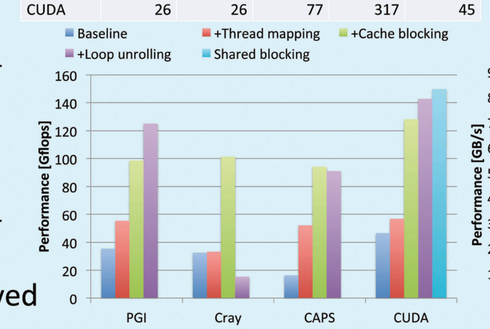
- Port and optimize each application
- Compare the performance and usability of OpenACC with CUDA

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

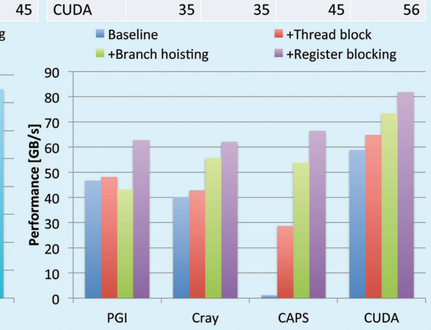
Matrix Multiplication
(Table: # of modified lines, Graph: Performance [Gflops])

	Baseline	Thread mapping	Cache blocking	loop unrolling	Shared blocking
OpenACC	9	11	62	302	
CUDA	26	26	77	317	45



7-Point Stencil
(Table: # of modified lines, Graph: Performance [GB/s])

	Baseline	Thread mapping	Branch Hoisting	Register blocking
OpenACC	7	10	18	29
CUDA	35	35	45	56

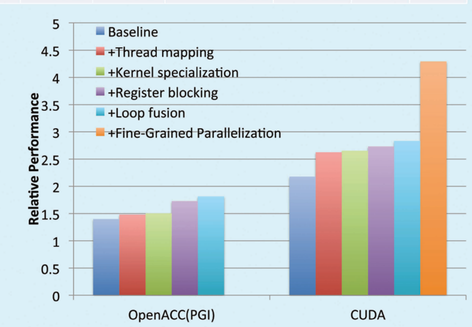


A real world CFD application

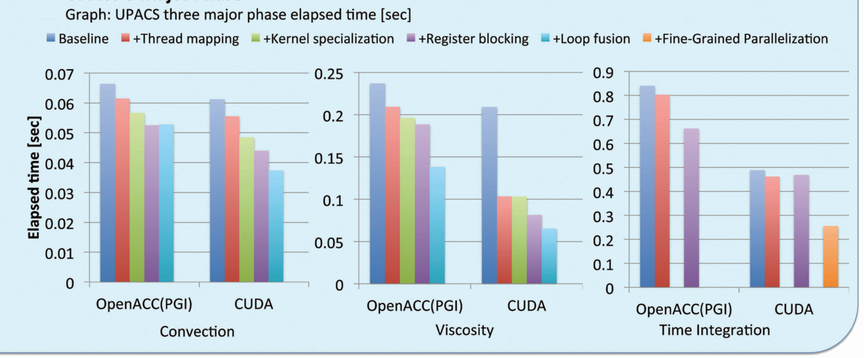
- Problem size: 120^3
- OpenACC performance
  - Baseline: achieved 64% of CUDA
  - Fully-optimized: acheived 42% of CUDA
- Fine Grained Parallelization using shared memory is the most efficient optimization
  - But using shared memory is limited in OpenACC, it causes the performance gap

UPACS
(Table: # of modified lines,
Graph: Relative Performance compared with OpenMP using 6cores)

	Baseline	Thread mapping	Kernel specialization	Register blocking	Loop fusion	MFGS
OpenACC	1788	1809	2601	2940	3296	
CUDA	5607	5743	6614	7641	8656	8870



UPACS 3 Major Phase
Graph: UPACS three major phase elapsed time [sec]



Future Work

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