An OpenACC Extension for Data Layout Transformation

Tetsuya Hoshino†, Naoya Maruyama†, ††, Satoshi Matsuoka†

† Tokyo Institute of Technology
†† RIKEN Advanced Institute for Computational Science

Background and Motivation

Various Types of Supercomputers

- Multi-core CPU systems
  - K computer (4th, in top500 Jun. 2014)
- MIC systems
  - Tianhe-2 (1st, in top500 Jun. 2014)
- GPU systems
  - TSUBAME2.5(13th, in top500 Jun 2014)

Top500 List - June2014 available from : http://www.top500.org/

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Super Computer Center in Guangzhou China	Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3120000	33862.7	54902.4	17808
2	DOE/SC/Oak Ridge National Laboratory United States	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560640	17590.0	27112.5	8209
3	DOE/NNSA/LLNL United States	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1572864	17173.2	20132.7	7890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705024	10510.0	11280.4	12660
5	DOE/SC/Argonne National Laboratory United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786432	8586.6	10066.3	3945
6	Swiss National Supercomputing Centre (CSCS) Switzerland	Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115984	6271.0	7788.9	2325
7	Texas Advanced Computing Center/Univ. of Texas United States	Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462462	5168.1	8520.1	4510
8	Forschungszentrum Juelich (FZJ) Germany	JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458752	5008.9	5872.0	2301

— Multi-core CPU — MIC — GPU

Programming Models for Accelerators

- CUDA･OpenCL
  - The most widely used programming interface for GPGPU
  - Low-level programming is required
- OpenACC
  - A new accelerator programming interface
  - High-level programming model (OpenMP-like directive-based)

	Performance	Programmability	Portability (Functional)	Portability (Performance)
CUDA	Very high	Low	Low	Low
OpenCL	Very high	(Very) Low	High	Low
OpenACC	High	High	High	Low

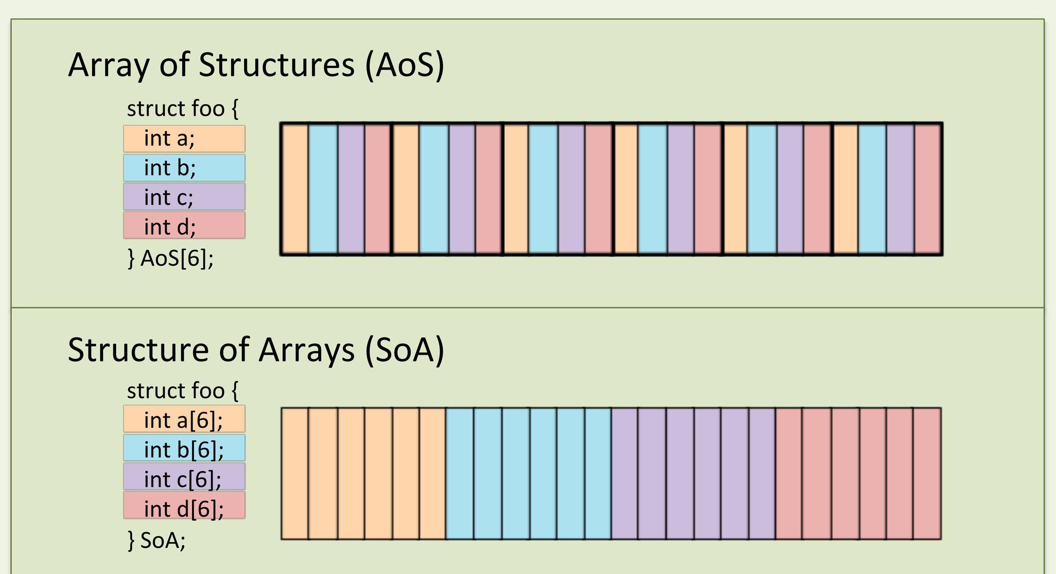
Problem: Performance Portability

Motivation

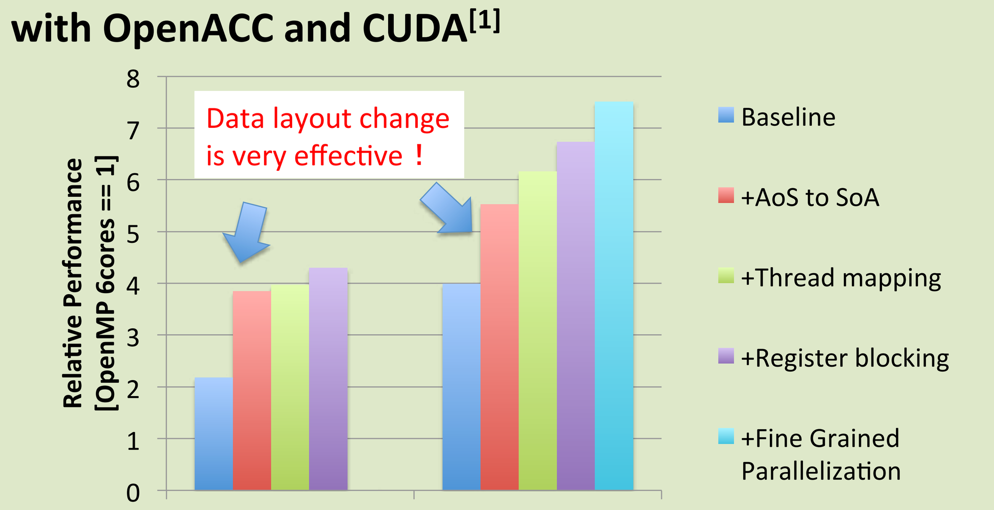
- To improve the performance portability of OpenACC

Performance Portability Problem: Data Layout

Data layout change is effective optimization for accelerators but it is also one of the cause of the low performance portability.



Porting and optimizing of a real world CFD application with OpenACC and CUDA[1]



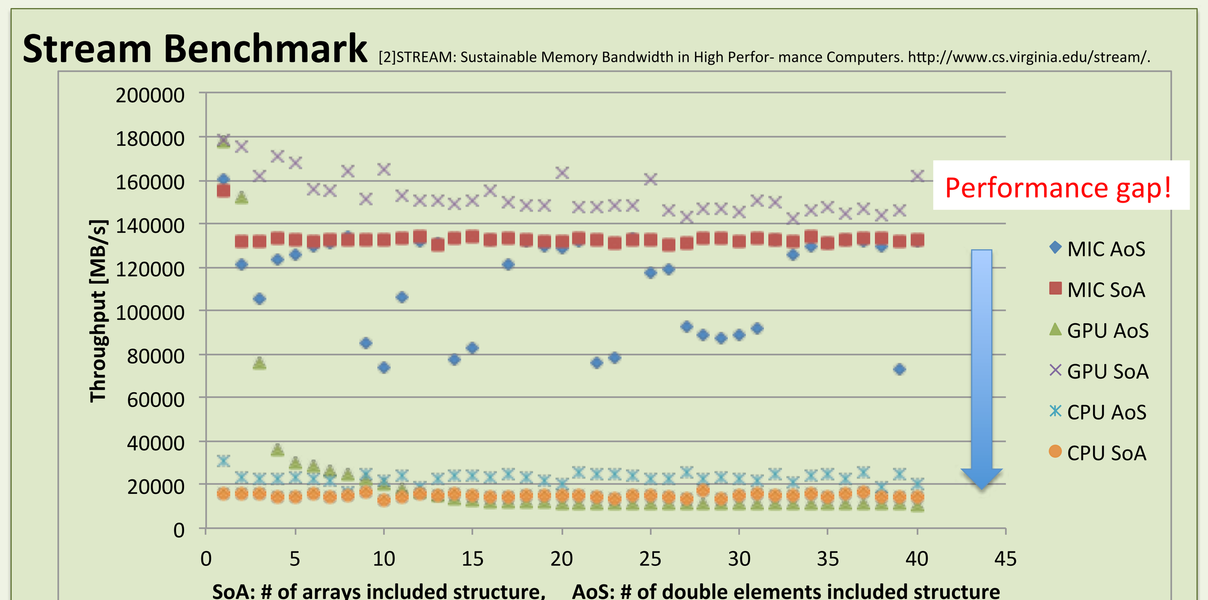
[1]Tetsuya Hoshino, Naoya Maruyama, Satoshi Matsuoka, Ryoji Takaki, "CUDA vs OpenACC: Performance Case Studies with Kernel Benchmarks and a Memory-Bound CFD Application," ccgrid, pp.136-143, 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2013

Benchmarks

- Stream Benchmark[2]
  - Changed data layout
  - Triad:

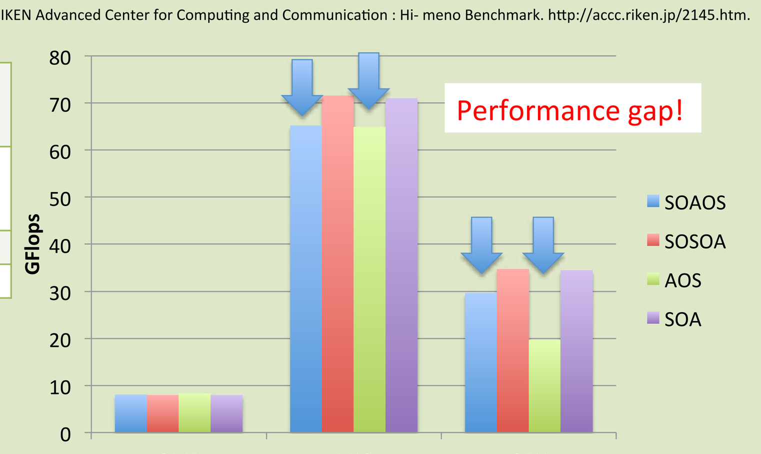
    $A = B + scalar \times C$
  - Double precision
- Himeno Benchmark[3]
  - Changed data layout of the coefficient matrices A, B and C
  - Single precision
  - Not fully optimized
    - Fully optimized CUDA results achieves over 90GFlops

Stream Benchmark [2]STREAM: Sustainable Memory Bandwidth in High Perfor- mance Computers. http://www.cs.virginia.edu/stream/.



Himeno Benchmark [3] RIKEN Advanced Center for Computing and Communication : Hi- meno Benchmark. http://accc.riken.jp/2145.htm.

SOAOS (Original)	a[MIMAX][MJMAX][MKMAX][4] b[MIMAX][MJMAX][MKMAX][3] c[MIMAX][MJMAX][MKMAX][3]
SOSOA	a[4][MIMAX][MJMAX][MKMAX] b[3][MIMAX][MJMAX][MKMAX] c[3][MIMAX][MJMAX][MKMAX]
AOS	abc[MIMAX][MJMAX][MKMAX][10]
SOA	abc[10][MIMAX][MJMAX][MKMAX]



Proposal and Implementation

Three Types of Target Arrays

- There are three types arrays
  - Multi-dimensional array
  - 1-dimensional array
  - derived type array
- They are same data layout from the viewpoint of memory access
- They have different characteristics from the viewpoint of compiler

```
struct four_double{
  double a, b, c, d;
};
double A[Z][Y][X][3];
double B[Z*Y * X * 3];
struct four_double C[Z][Y][X];
```

Data Layout Transformation Directive

#pragma acc transform [*clause*[[,] *clause*] …] newline
structured block

clause list

- **transpose**(multi-dimensional-array[start:length][]…[]::[r1,r2,…,rN])
  - ex.) transpose (A[0:Z][0:Y][0:X][0:4]::[4,1,2,3])

    A[Z][Y][X][4] → A'[4][Z][Y][X]
- **redim**(1-dimensional-array[start:length]::[len1,len2,…,lenN])
  - ex.) redim (B[0:Z*Y*X*4]::[Z,Y,X,4])

    B[Z*Y*X*4] → B'[Z][Y][X][4]
- **expand**(derived-type-array[start:length])
  - ex.) expand (C[0:Z][0:Y][0:X])

    four_double C[Z][Y][X] → double C[Z][Y][X][4]
- **redim-transpose** (1D-array[start:length]::[len1,len2,…,lenN]::[r1,r2,…rN])
  - ex.) redim-transpose(B[0:Z*Y*X*4]::[Z,X,Y,4]::[4,1,2,3])

    B[Z*Y*X*4] → B'[Z][Y][X][4] → B''[4][Z][Y][X]

Implementation

- We make a translator on top of the Rose compiler Infrastructure
- Source-to-source (Extended OpenACC to OpenACC)

Input

```
#pragma acc transform transpose(foo_a [0:100][0:100][0:3],[1,3,2])
{
#pragma acc data copy (foo_a[0:100][0:100][0:3], foo_b [0:100][0:100][0:3])
  {
#pragma acc kernels
#pragma acc loop gang independent
    for(k = 0;k < 100;k++){
#pragma acc loop vector independent
      for(j = 0;j < 100;j++){
        for(i = 0;i < 3;i++){
          foo_b[k][j][i] = foo_a[k][j][i];
        }
      }
    }
  }
}
```

Output

```
#pragma acc trans transpose ( foo_a [ 0 : 100 ] [ 0 : 100 ] [ 0 : 3 ], [ 1, 3, 2 ] )
{
  double *foo_a_generated__1_3_2;
  foo_a_generated__1_3_2 = ((void *)(malloc(sizeof(double) * 100 * 100 * 3)));
  transpose_foo_a_1_3_2(((double *)foo_a_generated__1_3_2 ),((double *)foo_a));
#pragma acc data copy (foo_a_generated__1_3_2[0:100 * 100 * 3], foo_b[0:100][0:100]
[0:3])
  {
#pragma acc kernels
#pragma acc loop gang independent
    for (k = 0; k < 100; k++) {
#pragma acc loop vector independent
      for (j = 0; j < 100; j++) {
        for (i = 0; i < 3; i++) {
          foo_b[k][j][i] = foo_a_generated__1_3_2[((0 * 100 + k) * 3 + i) * 100 + j];
        }
      }
    }
  }
  retranspose_foo_a_1_3_2(((double *)foo_a), ((double *) foo_a_generated__1_3_2));
  free(foo_a_generated__1_3_2);
}
```

Performance Evaluation

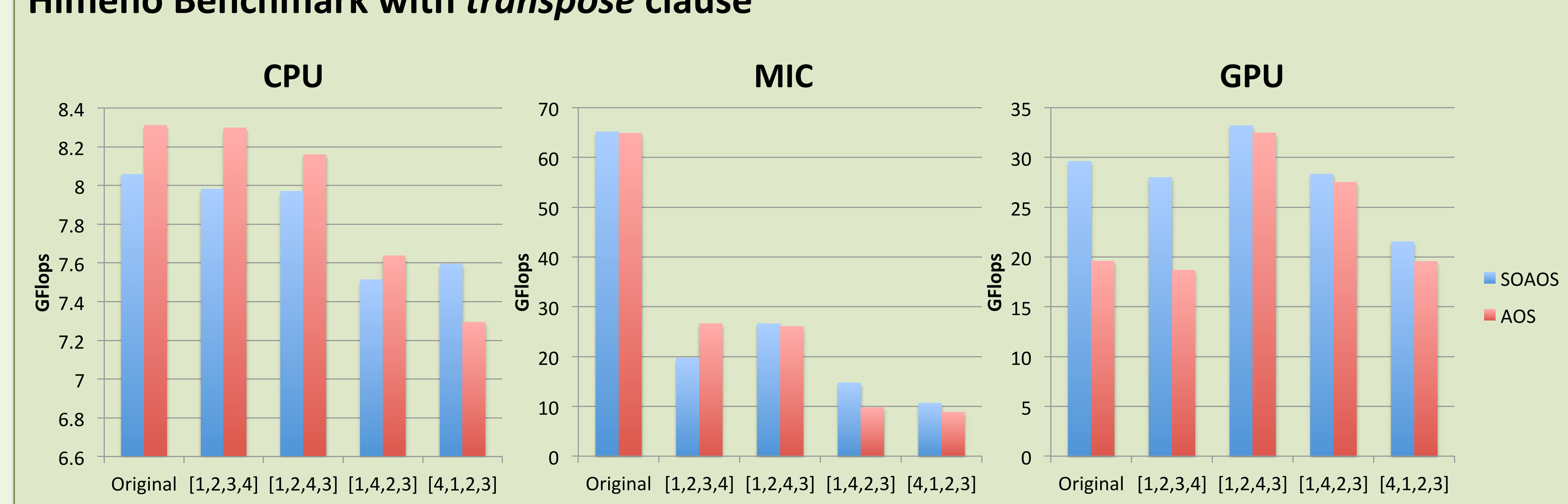
Evaluation with Himeno Benchmark

- We Implement *transpose* clause and apply to the coefficient matrices of Himeno Benchmark
  - The x-axis of the graph is the rule of transpose
  - Transposed with [1,2,3,4] is the same data layout of the original one
- Our translator achieves as much as 165% in performance for GPUs compared with the data layout which is the best for CPUs

Evaluation Environment

	Environment	Compilers	Options
CPU	Intel Xeon X5670 6cores 2.93GHz 2sockets 54GB Memory	icc 14.0.2	-O3 -openmp
GPU	NVIDIA Kepler K20X 2688 CUDA cores 6GB Memory	pgcc 14.2	-O3 -ta=nvidia,cc35,kepler
MIC	Intel Xeon Phi 7120X 61 cores 16GB Memory	icc 14.0.2	-O3 -mmic -openmp -opt-prefetch-distance=4,1 -opt-streaming-stores always -opt-streaming-cache-evict=0

Himeno Benchmark with *transpose* clause



Future Work

- Support all clauses
- Support more complicated data layout
- Implement auto-tuning mechanism between different devices
- Evaluate with real world applications