OpenACC Implementation of Radiative Transfer Simulation Code

Ryohei Kobayashi[1,2], Norihisa Fujita[1,2], Yoshiki Yamaguchi[2,1], Taisuke Boku[1,2], Kohji Yoshikawa [1,3], Makito Abe[1,3] and Masayuki Umemura[1,3]

1: Center for Computational Sciences, University of Tsukuba, Japan
2: Faculty of Engineering, Information and Systems, University of Tsukuba, University of Tsukuba, Japan
3: Graduate School of Pure and Applied Sciences, University of Tsukuba, Japan

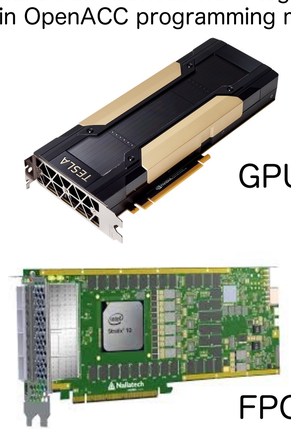
❖ Introduction

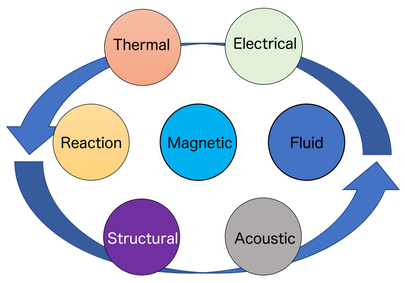
Motivation

- Realizing a way to easily use GPU and FPGA by OpenACC
  - Why OpenACC?
    - ✓high maintainability
    - ✓high portability
    - → quite friendly to application developers!
- Why GPU-FPGA coupling is needed?
  - Keyword: Multiphysics
  - Simulations with multiple interacting physical properties
    - ✓space – particle reaction
    - ✓fluid dynamics with chemical reaction
    - ✓macroscopic/microscopic hybrid simulation for molecular dynamics
  - Various computations are included within a simulation
    → Hard to accelerate simulation speed by GPU only

We aim to combine GPU and FPGA to improve simulation speed (offloading GPU non-suited computation part to the FPGA)

Make both devices work together in OpenACC programming model

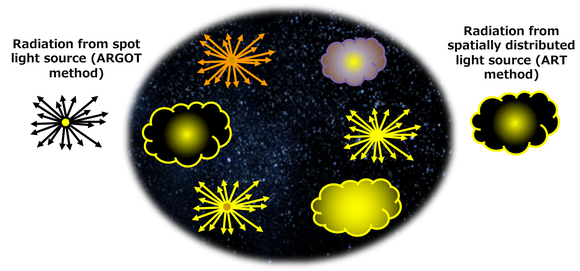




Schematic representation of Multiphisics

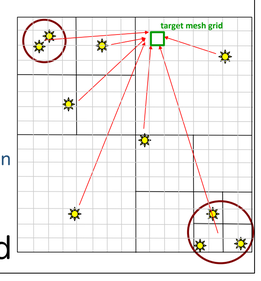
What we have to do first is…

- Investigating the performance change by converting to OpenACC
  - we implemented ARGOT code with OpenACC and evaluated its performance by comparing with those of OpenMP-based CPU implementation and CUDA-based GPU implementation
- ARGOT code: an astrophysics simulation code
  - developed in Center for Computational Sciences, University of Tsukuba
  - combining two types of radiative transfer
    - ✓ARGOT method
      - a radiative transfer from a spotlight (point source)
    - ✓ART method
      - a radiative transfer from spatially distributed light
  - CPU (OpenMP) and GPU (CUDA) implementations are already available



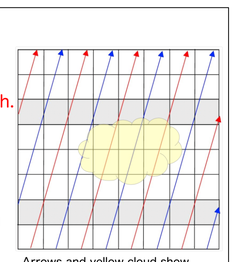
Our target application: ARGOT code (one of the Multiphisics simulations)

- is to solve the radiative transfer from point radiation sources
- builds an oct-tree data structure for the distribution of radiation sources
  - It's a 3-dimensional problem space, so it's an octave tree
  - For point of view from the target mesh , it is treated as a same light source if the light sources fall within a certain degree
    effective number of sources : N_s ➡ $\log N_s$
- is similar to Tree-Code in gravity calculation
  - suitable for the GPU
  - Each ray is assigned to each CUDA thread, and then these computations are conducted in parallel



ARGOT method

- ART method is based on ray-tracing method
  - 3D target space split into 3D meshes
  - Rays come from boundaries and move in straight in parallel with each other
- This method is not suitable for SMID-style arch.
  - Memory access pattern for mesh data is varies depending on ray's direction
    - ✓causing a decrease in cash hit ratio
  - Atomic operations are needed
    - ✓for processing the superimposition of the effects of multiple rays on the shaded mesh
    - ✓To avoid atomic operations, ray tracing along the red and blue light must be separately performed
      - causing the memory access patterns to become more scattered



Arrows and yellow cloud show rays and gas to compute reactions, respectively

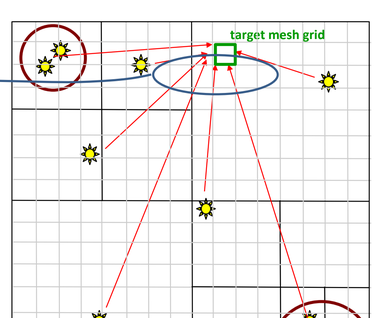
ART method

❖ OpenACC implementation

Loop optimizations

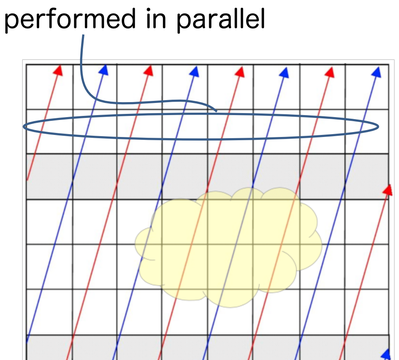
- Parallel regions are specified like OpenMP
  - Replace #pragma omp parallel for with #pragma acc parallel loop
    - ✓The CUDA implementation is based on the OpenMP implementation, so the parts that are offloaded to the GPU should be equivalent.
    - ✓Function calls in parallel regions should be OpenACCed with #pragma acc route seq.
- Add asnyc clause for asynchronous execution
- The parallel granularity is set based on the block size and number of threads of the CUDA implementation
  - Set num_gang and vector_length

Each ray is mapped to a thread, and radiative transfer is performed in parallel



Each ray is mapped to each thread, and radiative transfer from point sources is performed in parallel.

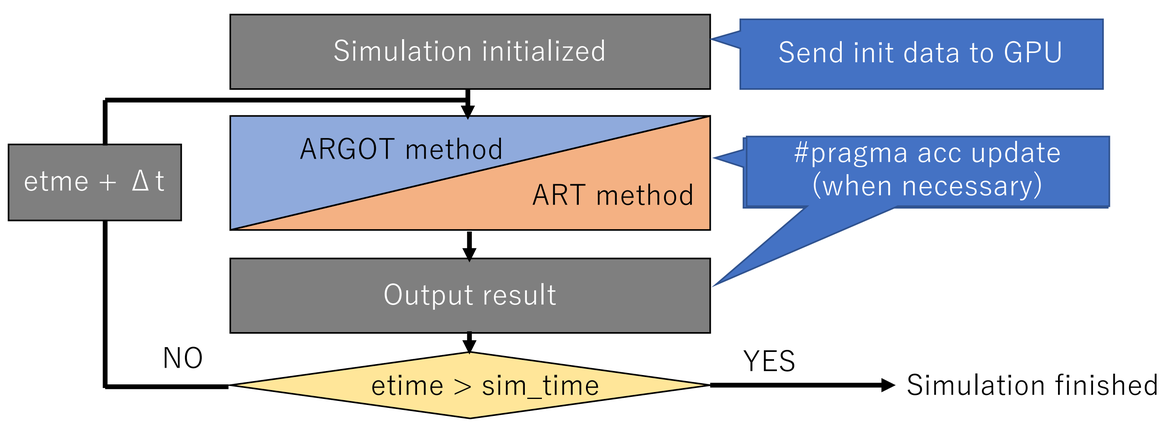
ARGOT method



ART method

Data management

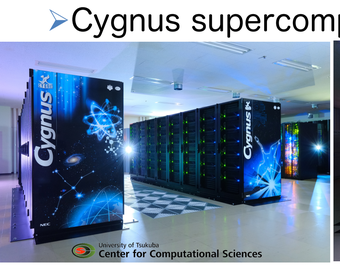
- Data transfer is implemented to be equivalent to the CUDA implementation
  - At initialization, use #pragma acc enter data copyin() to send data to GPU memory
    - ✓#pragma acc update host() is used only for processes that need to be run on the host (pulling the necessary data from the GPU)
    - ✓#pragma acc update device() is used to update the GPU memory with the host's updated data
  - As with Loop optimizations, async clause is added to asynchronous execution

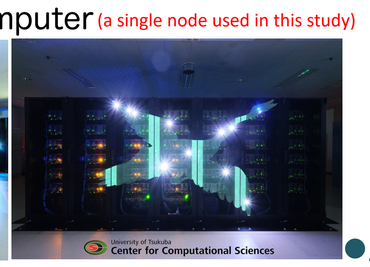


Simulation flow of ARGOT code

❖ Performance comparison

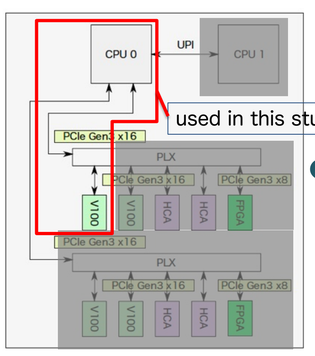
- Evaluation testbed
  - Cygnus supercomputer (a single node used in this study)

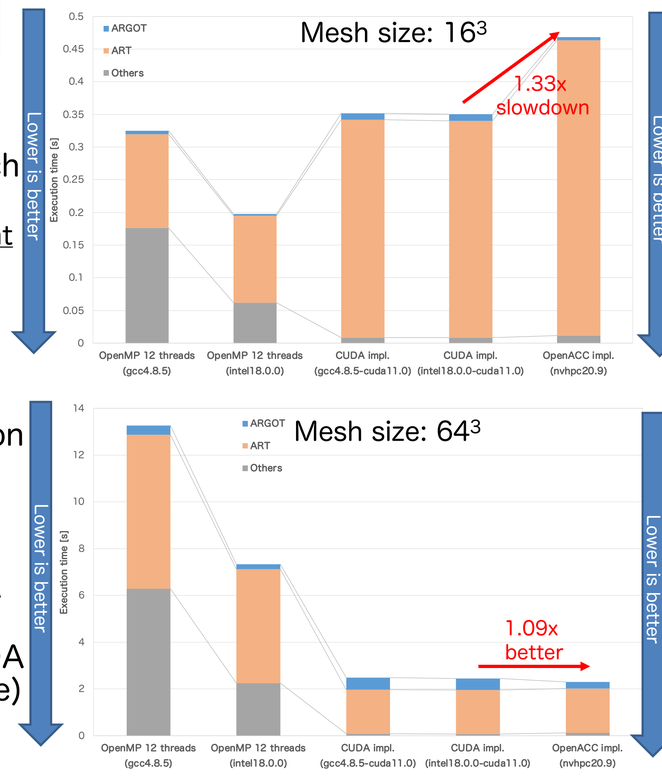


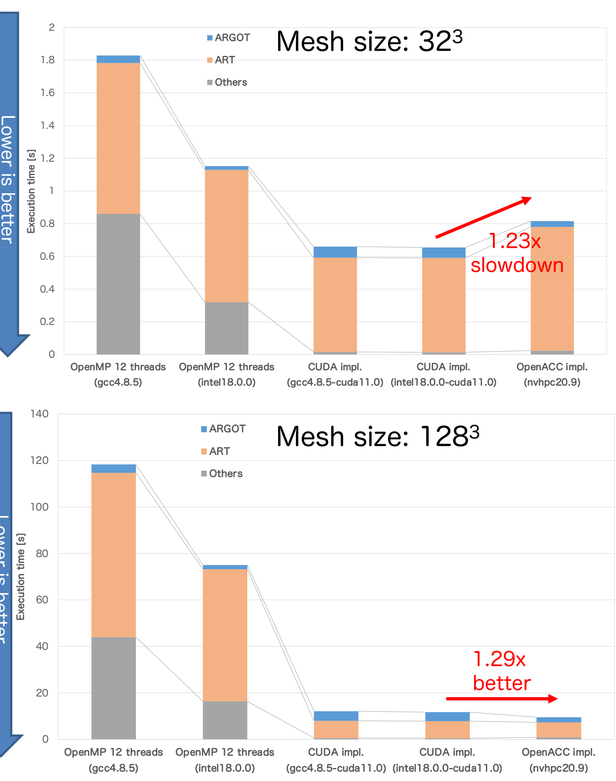


- Others
  - solving chemical reactions and the radiative heating and/or cooling of each mesh
    - ✓Computation of a mesh highly independent from one another, which is why the GPU implementation can accelerate their execution
- ARGOT method
  - The larger the mesh size, the more noticeable the performance degradation of the CUDA impl. (this is because of cudahostalloc)
- ART method
  - Small mesh size: 73% ~ 76% of CUDA impl. Performance
  - Large mesh size: almost same as CUDA impl. (1.1x better when 128^3 mesh size)

Hardware specification	
CPU	Intel Xeon Gold 6126 x 2
Host Memory	192 GiB (16 GiB DDR4-2666 ECC RDIMM x 6 chan. x 2)
GPU	NVIDIA Tesla V100 (PCIe Gen3 x16) x 4
GPU Memory	32 GiB CoWoS HBM2 @ 900 GB/s with ECC
Software specification	
OS	CentOS 7.9
Host Compiler	gcc 4.8.5, Intel compiler 18.0.0 (for OpenMP impl.)
CUDA Toolkit	CUDA 11.0.221 (for CUDA impl.)
OpenACC Compiler	NVIDIA HPC SDK 20.9







❖ Conclusion and future work

- What we did is...
  - as a preliminary evaluation for the realization of GPU-FPGA integration with high usability, we implemented the ARGOT code by OpenACC and evaluated the performance of OpenMP-based CPU implementation and CUDA-based GPU implementation
  - we confirmed that...
    - ✓For small mesh size: OpenACC-based implementation achieved up to 77% of the performance CUDA-based ARGOT code implementation
    - ✓For large mesh size: OpenACC-based implementation achieved up to 1.29x better performance compared to CUDA-based ARGOT code implementation
- Next step is...
  - to evaluate multi-node versioned OpenACC implementation's performance by comparing to OpenMP and CUDA implementations

ACKNOWLEDGEMENT
This work was supported in part by the "Next Generation High-Performance Computing Infrastructures and Applications R&D Program" (Development of Computing Communication Unified Supercomputer in Next Generation) of MEXT. This research was also supported in part by the Multidisciplinary Cooperative Research Program in CCS, University of Tsukuba, and JSPS KAKENHI, Grant Number 21H04869. We also thank Dr. Naruhiko Tan of NVIDIA for his advice on OpenACC optimization.