THE UNIVERSITY OF TOKYO

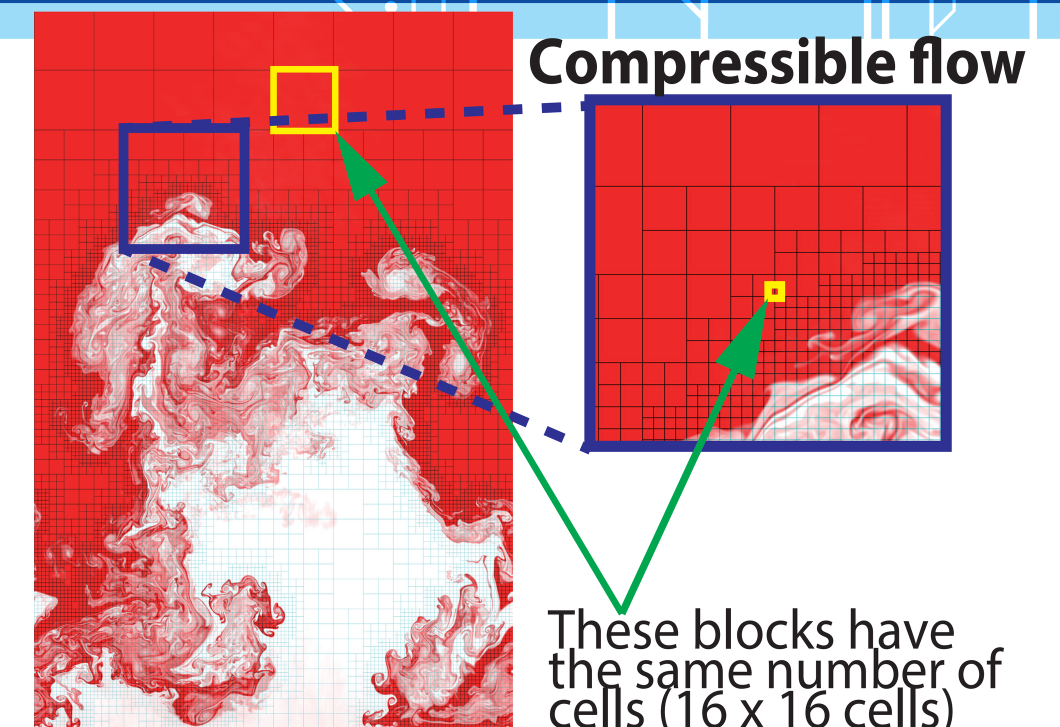
東京大学情報基盤センター INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

JCAHPC

AMR framework for multiple GPUs

Takashi Shimokawabe, shimokawabe@cc.u-tokyo.ac.jp

Adaptive mesh refinement (AMR)



Recently grid-based physical simulations with GPU require effective methods to adapt grid resolution to certain sensitive regions of simulations. An adaptive mesh refinement (AMR) method is one of the effective methods to compute certain local regions that demand higher accuracy with higher resolution. The AMR method on the GPU is, however, complicated and requires the high development cost.

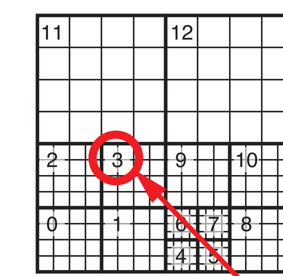
AMR Framework for GPU computing

We are currently developing an AMR framework for realizing effective high-resolution simulations on GPU. The proposed framework is implemented in the C++ language and automatically translates user-written functions that update a grid point and generates both GPU and CPU code.

AMR data structure

The AMR framework uses Octree for 3D and Quadtree for 2D to represent spatial distribution of decomposed grids. In order to achieve high performance on GPU, physical values on all leaves of the trees are actually stored in a single array allocated as contiguous memory area. The single array is decomposed as multiple "blocks" , each of which typically contains 16 x 16 cells in 2D with halo regions. Each leaf of the trees just holds an ID that indicates the assigned block on the array.

Tree structure



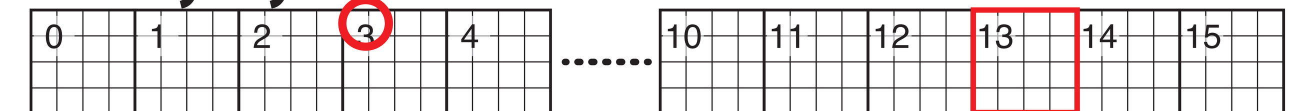
Level 0 1 2 3

a b c 11 12 0 1 2 3 d 8 9 10 4 5 6 7

c Branch
3 Leaf

ID mapping

Memory layout



A block (16 x 16 cells)

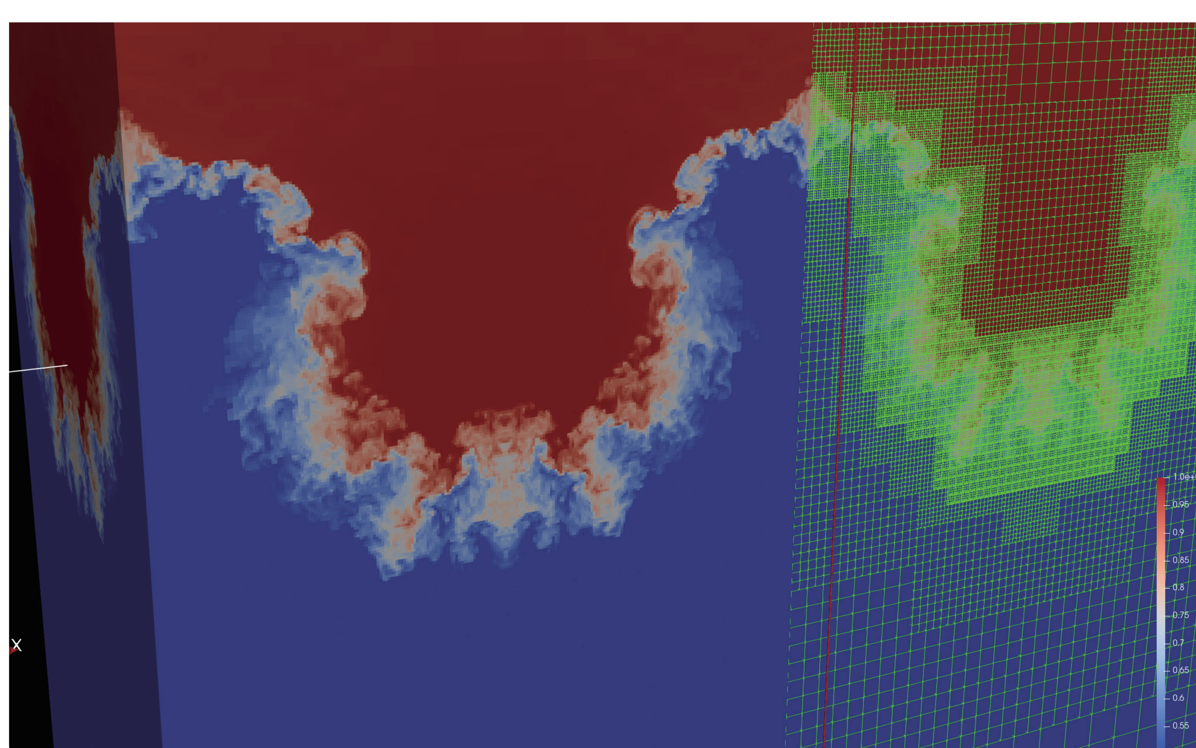
Writing and Executing Stencil Computation

To use this framework, programmer just provides "stencil functions" that update a grid point, which are defined as C++ functors. Since the framework is based on block-based AMR, stencil functions for Cartesian grid can be also used for AMR. Thanks to the data structure described above, the framework can easily execute given stencil functions over multiple blocks by just using a single CUDA kernel even when each block represents different resolution. This contributes to achieving higher performance on GPU.

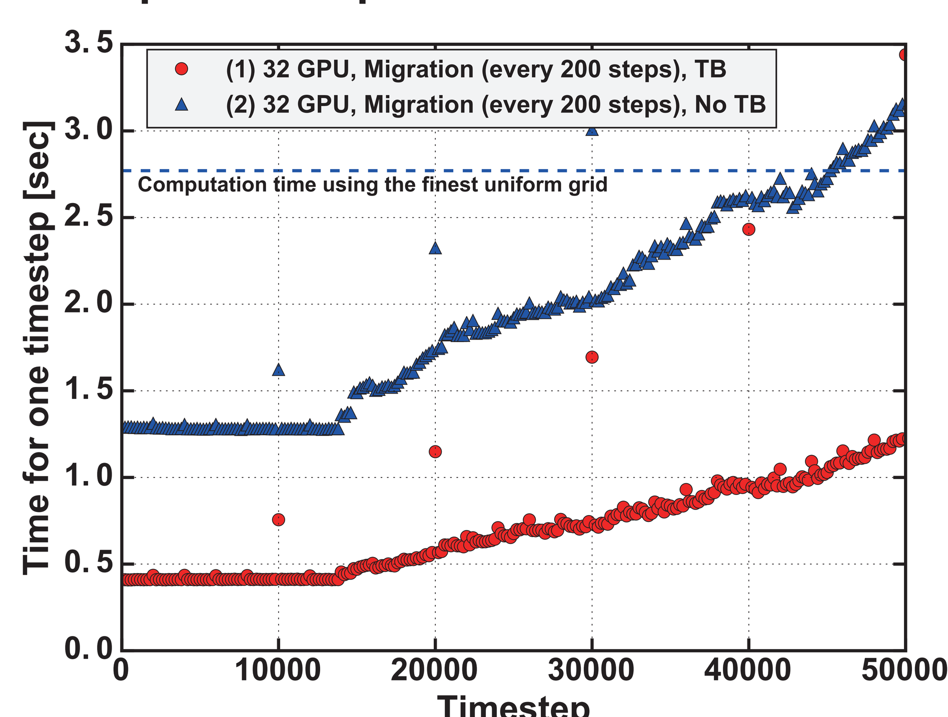
```
auto diffusion3d = [=] __host__ __device__
(const MArrayIndex &idx, int level,
 float ce, float cw, float cn, float cs, float ct,
 float cb, float cc, const FLOAT *f, float *fn) {
    fn[idx.ix()] = + cc*f[idx.ix()]
      +ce*f[idx.ix(1,0,0)]+cw*f[idx.ix(-1,0,0)]
      +cn*f[idx.ix(0,1,0)]+cs*f[idx.ix(0,-1,0)]
      +ct*f[idx.ix(0,0,1)]+cb*f[idx.ix(0,0,-1)];
}};
```

Rayleigh-Taylor instability simulation using AMR

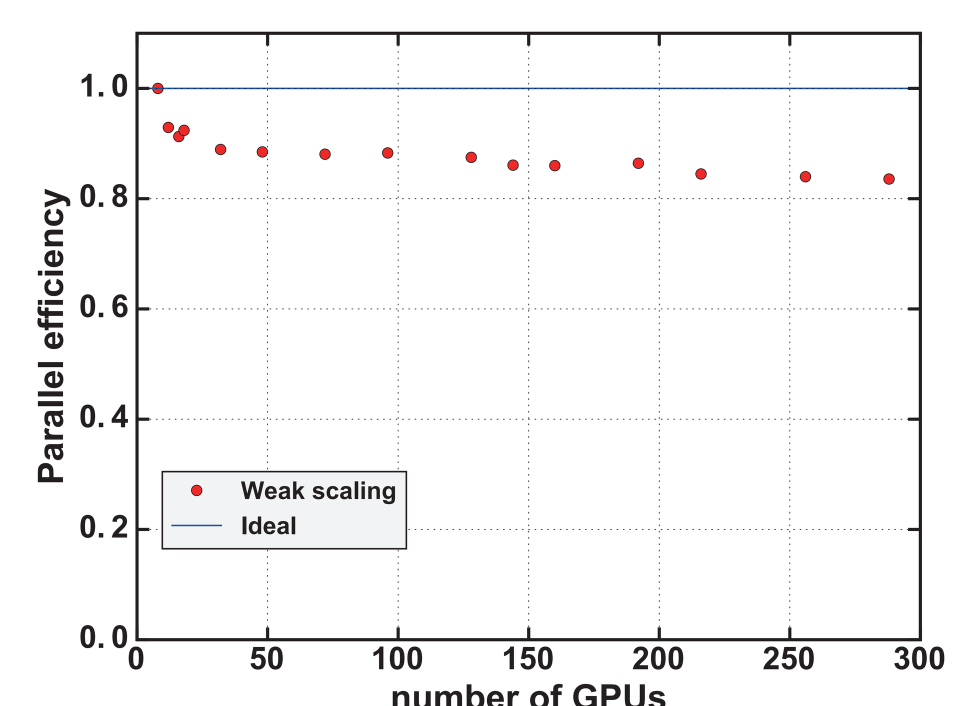
Left figure shows a snapshot of computational results of the Rayleigh-Taylor instability obtained by 3D compressible flow computation written by this AMR framework. By applying the AMR method to fluid simulation, we have succeeded in simulating with a fine structure around the interface of two fluids. This simulation uses 5 levels for AMR; width of the biggest cell is equal to 16 x width of the smallest cell.
Center figure shows the computation time taken for the calculation of each time step. It indicates that version (1), which exploits the temporal blocking (TB) method, is 6.7 times faster than the same computation on the finest uniform grid, which is depicted as a blue dashed line.
Right figure shows the weak scaling results of the simulation using 5-level AMR with the TB method and the data migration to improve load balancing. It has demonstrated good weak scalability with 84% of the parallel efficiency on multiple NVIDIA P100 GPUs on the TSUBAME3.0 supercomputer.



A snapshot of density distribution results obtained by the simulation of 3D compressible flow. The boundary lines of the grid blocks are also shown in part.



Computational times for each time step using 32 GPUs.



Weak scaling on TSUBAME3.0.

T. Shimokawabe et al., "A High-productivity Framework for Adaptive Mesh Refinement on Multiple GPUs," International Conference on Computational Science (ICCS) 2019.