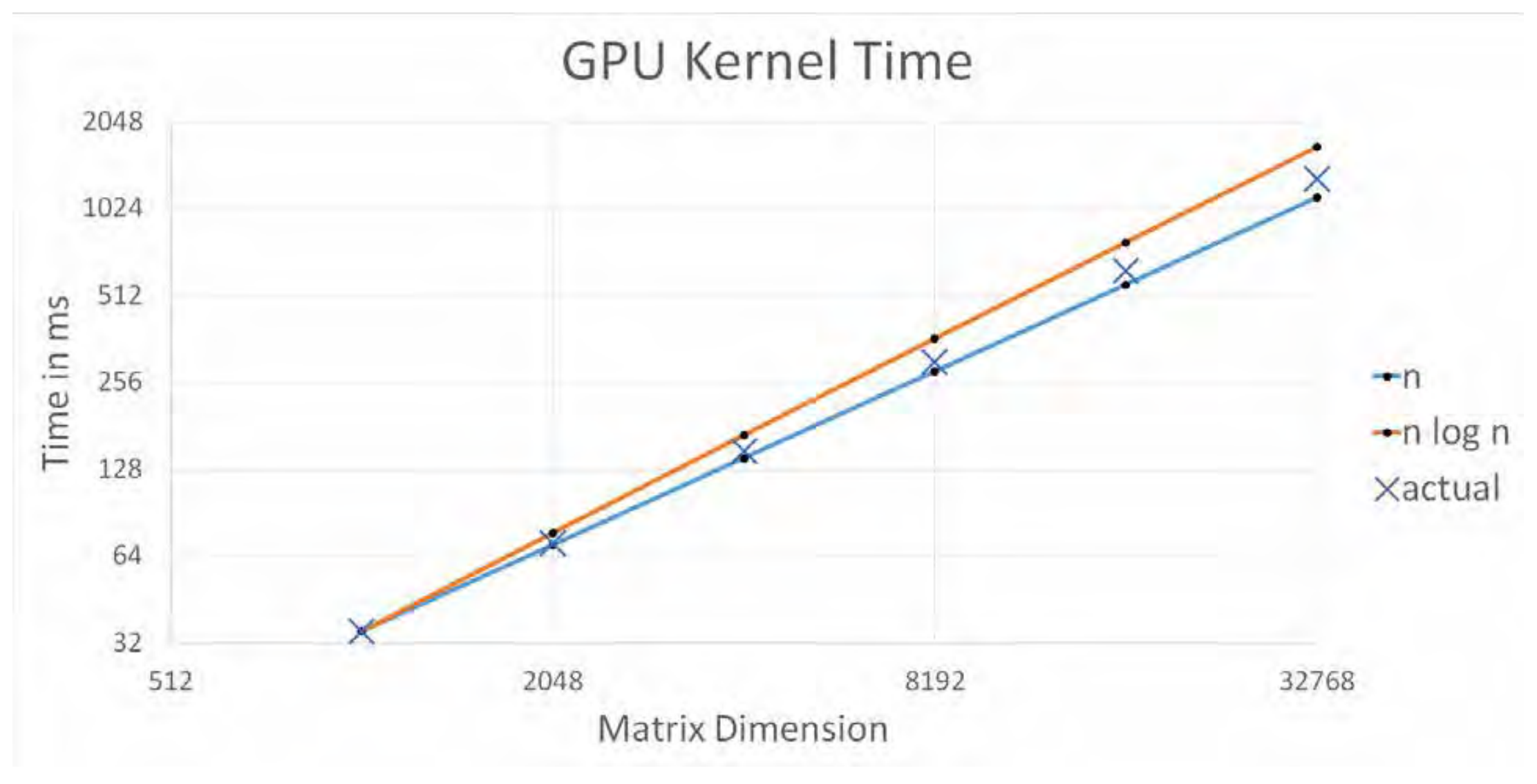
Applications on TSUBAME3.0

Fast Algorithms and Large Scale CFD

Fast Algorithms for HPC and Deep Learning

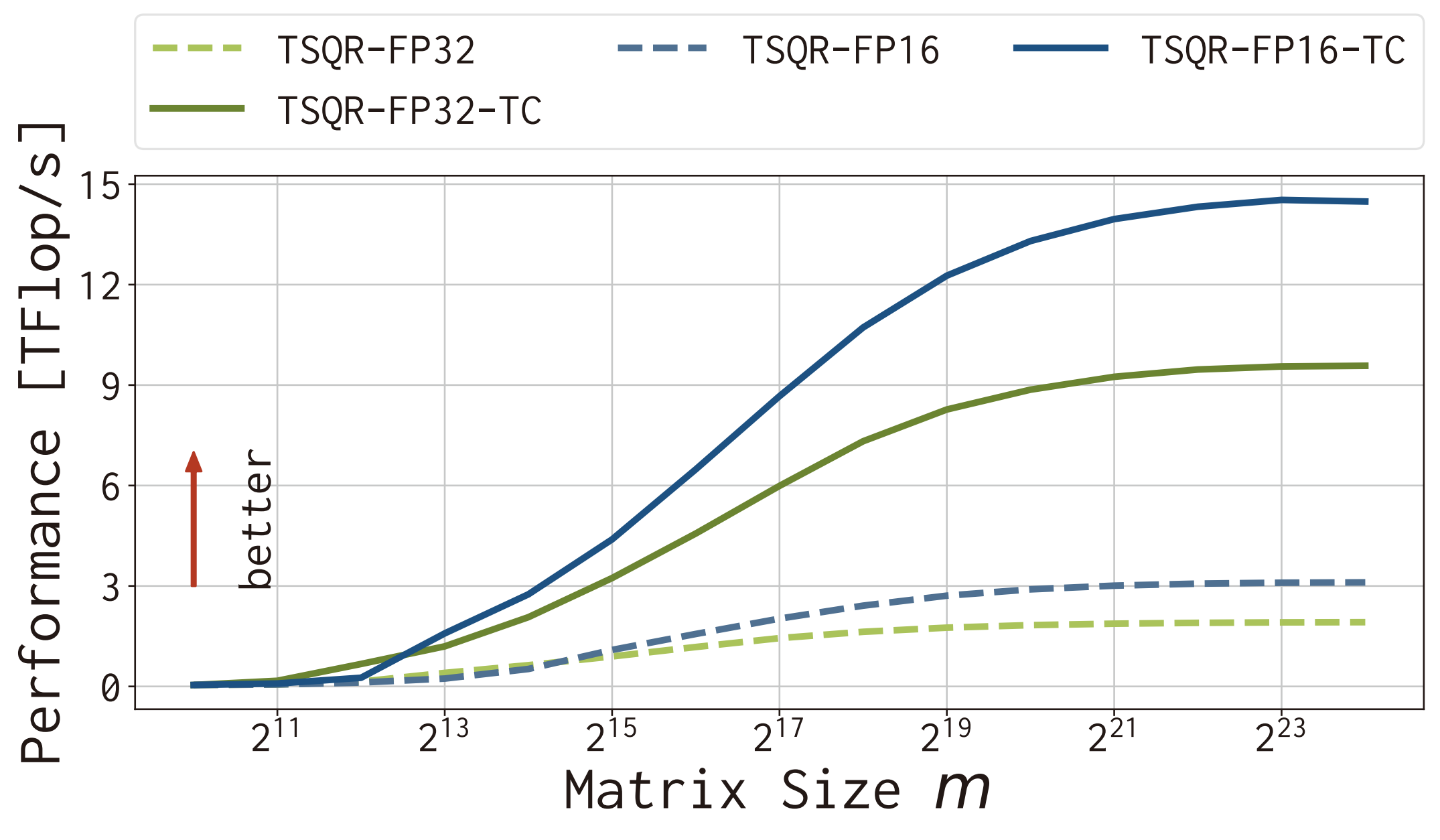
Data-driven H-matrix Factorization on GPUs



Matrix factorization is difficult to parallelize due to its dependency between the blocks. This difficulty is amplified by load-balancing issues when some of the blocks are low-rank. It is even more complicated when the blocks are hierarchically subdivided.

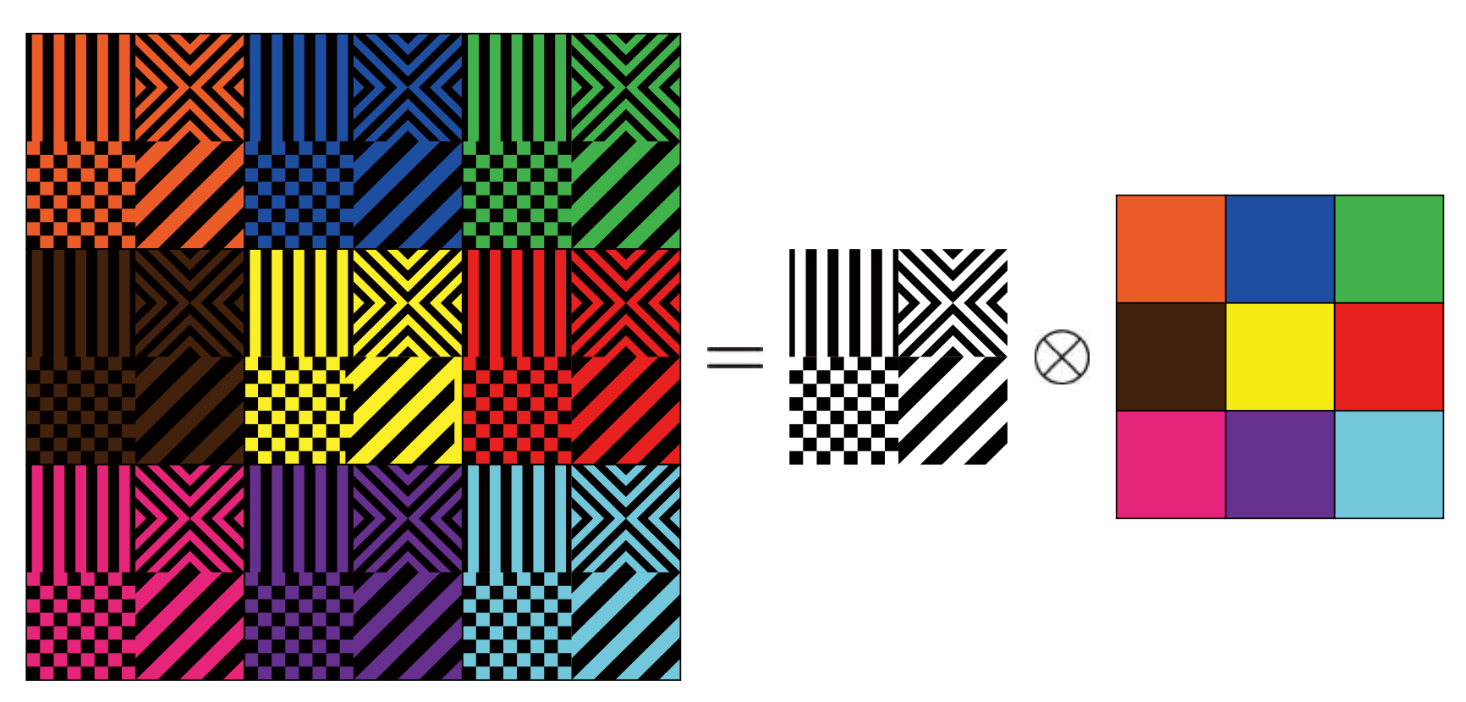
We tackle this challenging problem of H-matrix factorization and port the code to GPU. We developed our light-weight scheduler because existing ones like StarPU have too much overhead at the granularity we want to use it. Our implementation shows, for the first time, an ideal O(N) scalability on GPUs.

TSQR on TensorCores



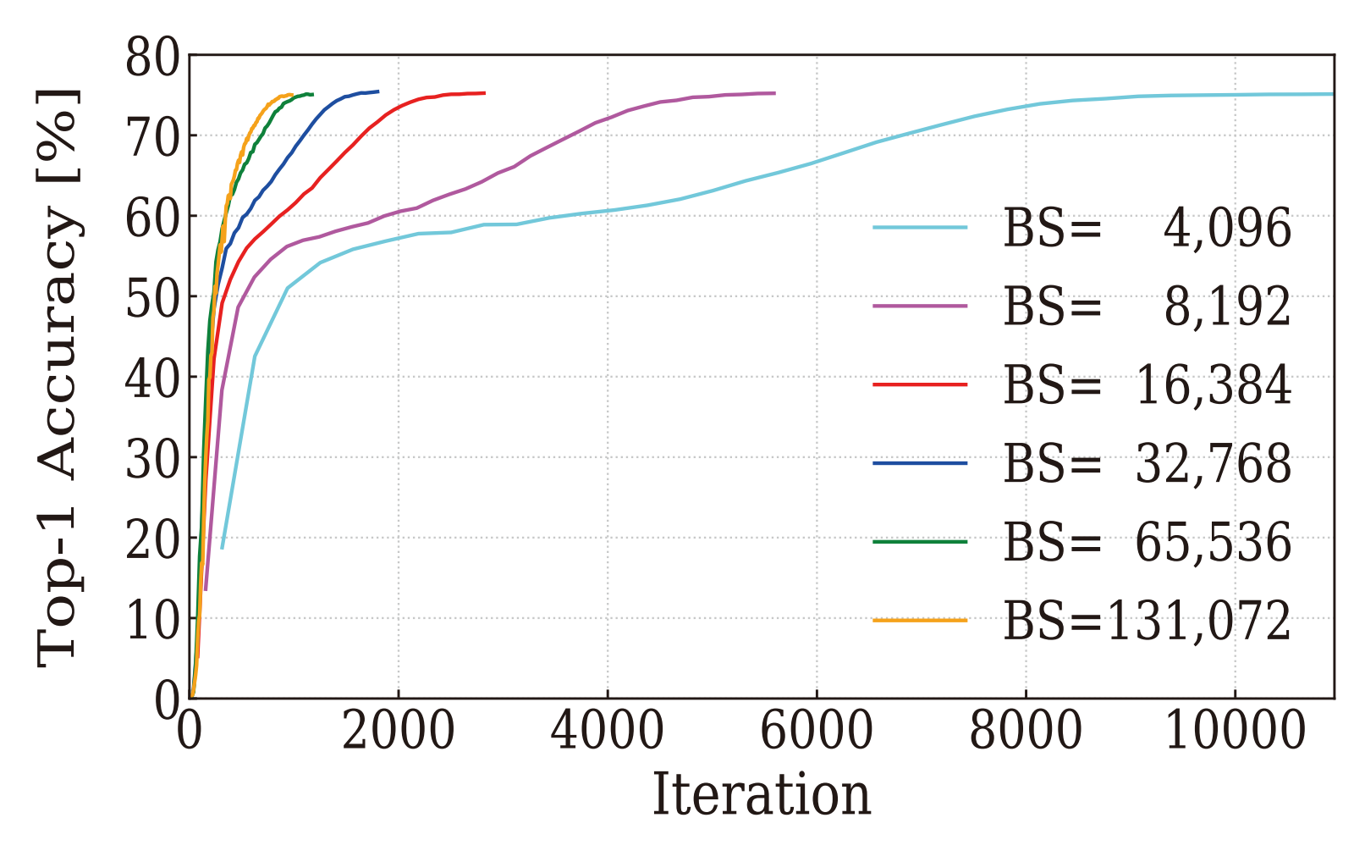
The compression of H-matrices involves a QR decomposition of tall and skinny matrices --TSQR. Since the goal of the compression is to perform a low-rank approximation of the original dense matrix, the accuracy required for this operation is not so high. This means that we can use the TensorCores on the latest GPUs to perform the TSQR much faster than usual. We investigate the amount of performance we can achieve when using different precision (FP16 and FP32) along with the TensorCore. We find that we can get close to 15 TFlops for tall matrices.

Kronecker Factorization in Deep Learning



The use of second order methods for optimizing deep neural networks could offer better convergence, but the calculation of the Hessian is prohibitively expensive. The use of Kronecker factorization for the Hessian matrix is a natural approach since it is constructed during back-propagation from the Kronecker product. By using Kronecker factorization, we are able to use second order optimization in deep learning with minimum overhead. Our distributed memory implementation switches to model-parallelism during the calculation of the Kronecker factors and further reduces the overhead to make the per iteration time competitive with SGD.

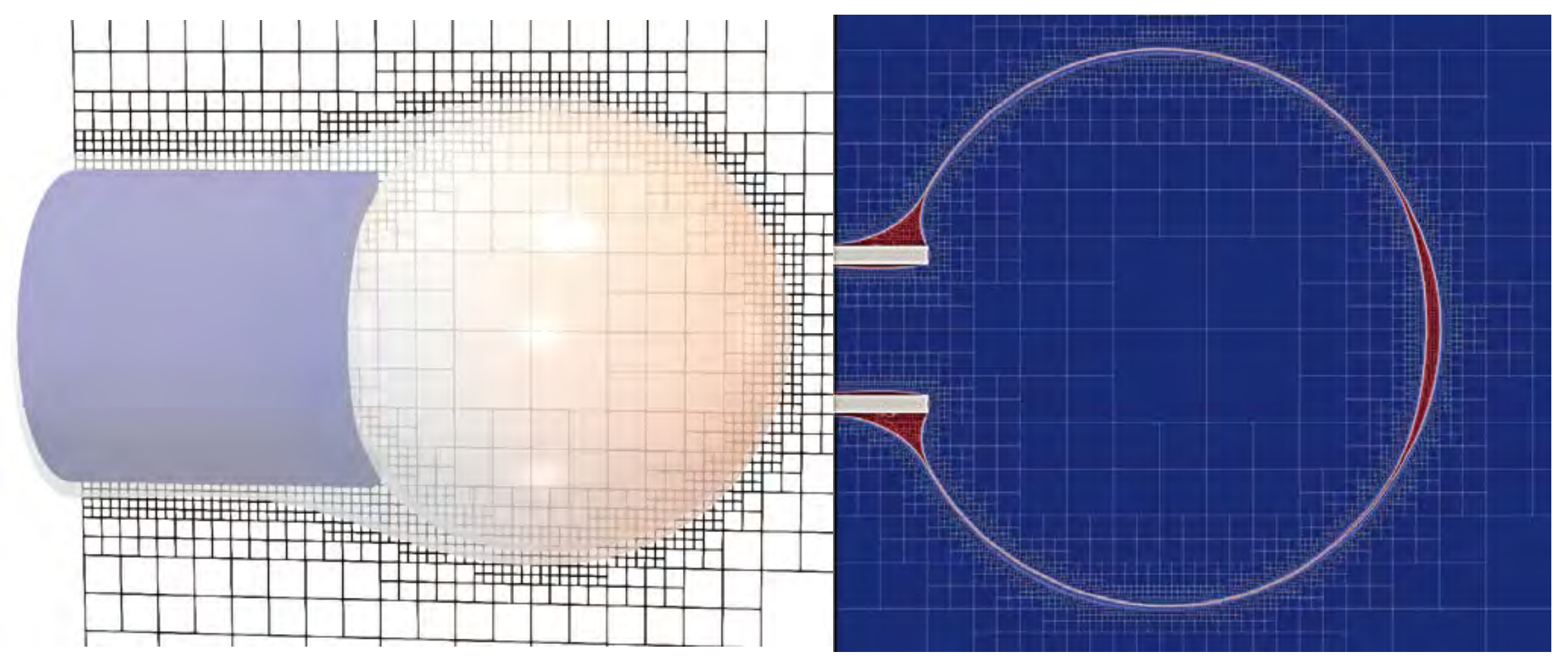
Training ImageNet in 2 minutes on 2048 GPUs



Data-parallel training of deep neural networks suffers from the large-batch problem, where the generalization gap increases as the mini-batch size increases proportionally to the number of GPUs. Using the Kronecker Factorization mention on the right, we are able to use a second order optimization method with minimum overhead, which allows us to retain the convergence even for extremely large batch sizes. We trained ImageNet in 2 minutes on 2048 GPUs, using a batch size of 131,072.

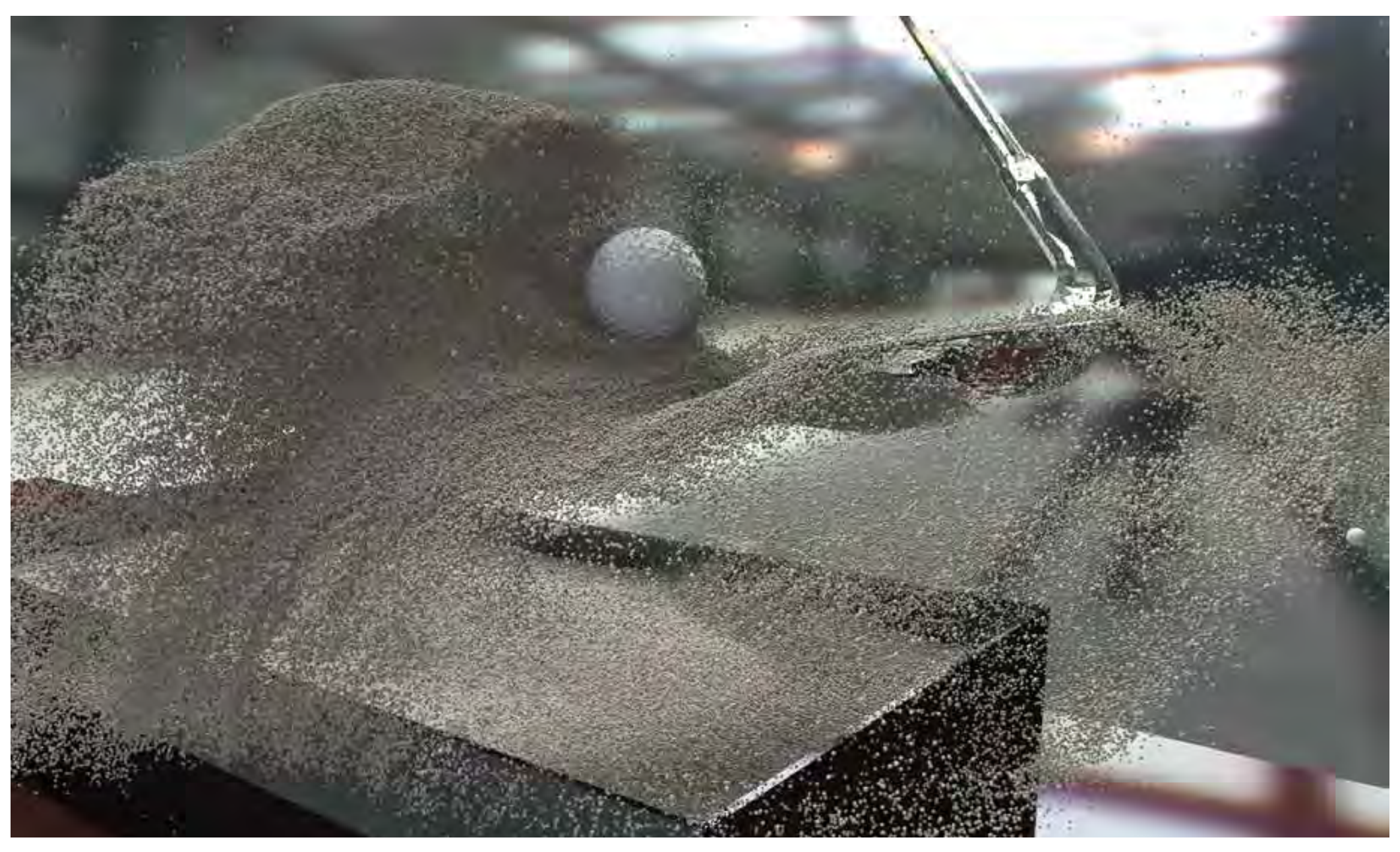
Large-scale Mesh-based and Particle-based Simulations

Weak-Compressible Flow Computations for Gas-Liquid Two-Phase Flows



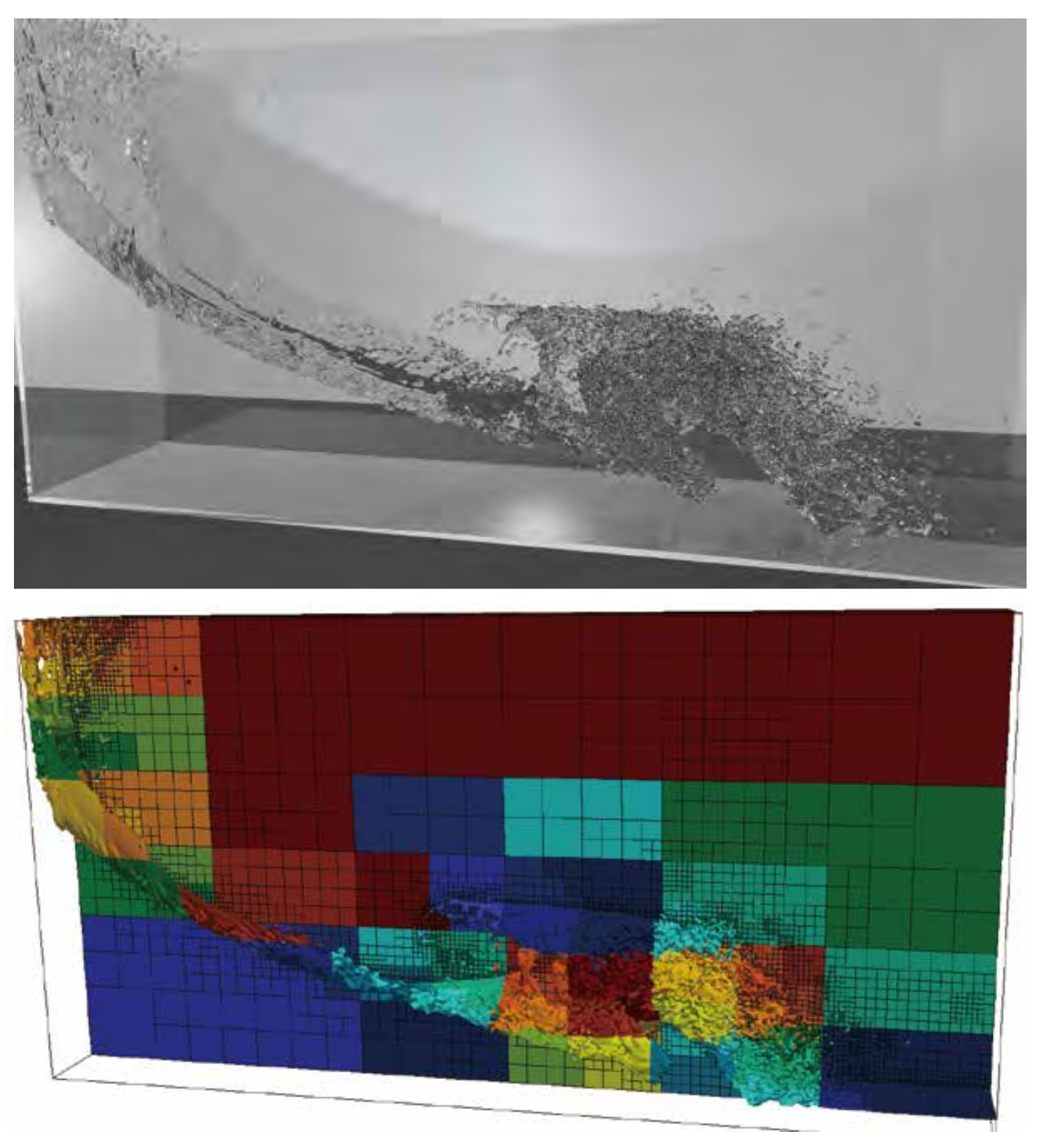
A weakly compressible scheme with an interface-adapted AMR method for incompressible gas-liquid two-phase flows without solving the Poisson equation has been developed. The fully explicit time integration is achieved by solving the pressure evolution equation derived from the compressible Naiver-Stokes equation under the condition of isothermal and low-Mach number. To reduce volume oscillation, the conservative Allen-Cahn equation coupled with Level-set method is solved. The results of benchmarks agree well with those of semi-implicit incompressible solvers and simulated 2D/3D soap bubble forming problems.

Large-scale Granular Simulation



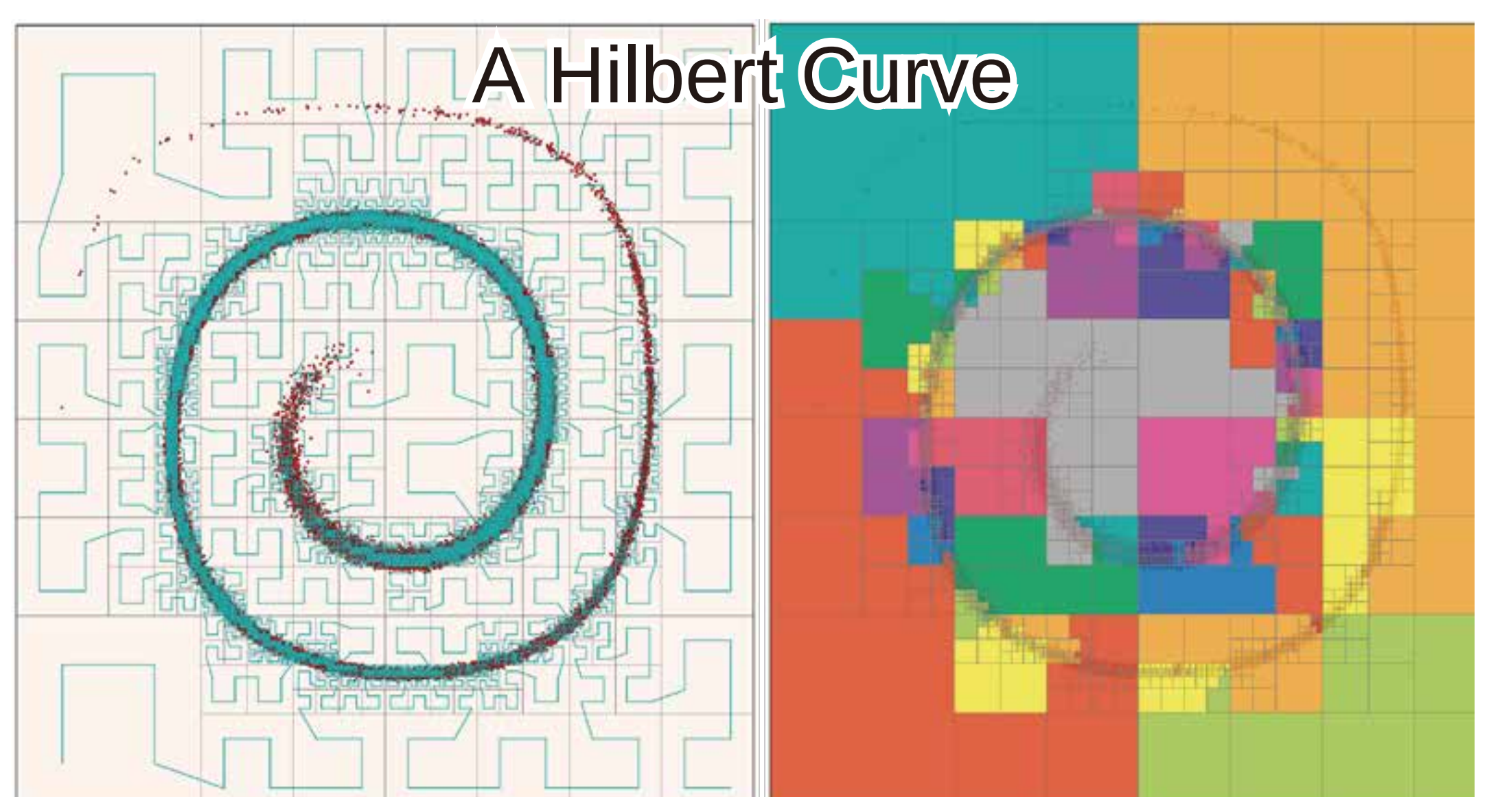
Discrete element method (DEM) is often used to simulate granular dynamics and its simple algorithm with the contact interaction is suitable for GPU computing. However, so many particles are included and the particle distributions are changing in time and space. A dynamic domain decomposition has to be introduced for multiple-node computing. In a bunker shot, the sand wedge does not hit the golf ball directly and transferring the force through the sand to the ball in order to reduce the impact. In this simulation, 16.7 million DEM particles are used to represent the dynamics of the sands with 256 GPUs.

Adaptive Mesh Refinement for Multi-phase Flows



Simulations for multi-phase flows require high-resolution grids to capture phenomena at the interface. By using the adaptive mesh refinement (AMR) method, which dynamically adapts high-resolution grids to interfaces, computational cost and memory usage are reduced. The spatial distribution of a computational load change in time; therefore, dynamic domain partitioning using a space-filling curve is introduced for multi-GPU computing to assign an equal number of grid points to each GPU. The figures show the large-scale free-surface flow simulations for the dam-breaking process and corresponding domain decomposition for 64 GPUs.

Dynamic Load Balancing using A Space-filling Curve



For large-scale particle-based simulation and Adaptive Mesh Refinement (AMR), it is a critical issue to achieve computational load balance and equal memory usage on multiple compute nodes. A domain partitioning in terms of a space-filling curve(SFC) is one of the promising candidates, and it recognizes a 1-dimensional mapping of 3-dimensional space by cutting with an equal length. Due to the low cost of SFC domain partitioning, it is suitable for frequent re-partitioning in the simulations of unsteady phenomena.