The 29th Computational Fluid Dynamics Symposium
Your Paper No.

Advanced GPU Direct-MPI Hybrid Framework with Block-Based Data Structure for Efficient AMR on Multi-GPU Systems

○ Un-Hong Wong, Tokyo Institute of Technology, 2-12-1 i7-3 O-okayama, Meguro-ku, Tokyo 152-8550,
Takayuki Aoki, Tokyo Institute of Technology, 2-12-1 i7-3 O-okayama, Meguro-ku, Tokyo 152-8550,

GPUs have been widely utilized in acceleration to scientific computations in recent years because of its high FLOPS (floating-point operations per second). However, due to the limited memory of a single GPU, the use of multi-GPU systems needs to be explored for large-scale simulations. GPUs provides extremely high computing power but the data communication is relatively slow which bottlenecks the performance of multi-GPU systems. In this paper, our GPU Direct-MPI hybrid framework for efficient data communication of large-scale grid-based simulation on multi-GPU systems is presented. A new block-based structured for efficient Adaptive Mesh Refinement (AMR) is also presented. An application of our framework to large-scale global magnetohydrodynamic (MHD) simulation of the solar wind interaction with Earth's magnetosphere is presented. The performance analysis of shows that using AMR with our framework saved 1.166 GB (25%) memory for each GPU with about 7% speedup for a computational domain of 512^3.

1. Introduction

Computing power is a critical resource in numerical simulation including computational fluid dynamic (CFD) and magnetohydrodynamic (MHD), especially at large-scale. Large-scale high resolution simulations require huge computational power and usually needs to be run on supercomputers or clusters for hours, days or even weeks to get the results. This is a consequence of the high resolution, complexity of the calculation during each step and because the simulation results are time dependent. Scientists and researchers are in need of a faster hardware or methods to speedup their simulation. This is the reason that general-purpose computing on graphics processing units (GPGPU) has become popular. Graphics processing units (GPUs) used to be the graphics acceleration hardware for boosting up the calculations in computer graphics. Due to its high parallelism, GPUs have been used as an accelerator and now play an important role in high performance computing (HPC). Thousands of researches using GPGPU to achieve fast computation results have been published in the past decade.

Data processed using a GPU had to be loaded to the dedicate memory of the GPU board — the device memory (GRAM). The capacity of the device memory on a single GPU is limited. Thus, multi-GPU systems are needed for large-scale simulations. However, the data transfer between multiple GPUs used to be done by a GPU-CPU-GPU approach — copy the data from the device memory of a GPU to host memory, and then to the device memory of another GPU. Therefore, the data communication between GPUs bottlenecks the efficiency of the simulations on multi-GPU systems. Several techniques and approaches were raised to address this problem including our GPU Direct-MPI parallel data communication (1)(2).

The resolution or size of the calculation domain of the numerical simulation depend by the amount of memory of the system. When the computational complexity is high, the resolution or calculation domain will be limited since more memory might be arranged for the use of the numerical scheme. In HPC, there is always a trade-off between memory and performance. In many cases, especially for parallel computing, users "pay" more memory to "gain" performance. For example, large-scale simulations running on supercomputer or cluster require halo grid points and some additional buffers for data communication, which is not necessary for a simulation running on a standalone machine. For multi-processes/multi-threading simulation code, each process/thread has to have its own memory space for the intermediate results (we call it workspace hereafter) in the calculation of the numerical scheme. Therefore, it requires more memory of each node than a single process/thread simulation code. Higher percentage of memory usage has to be allocated to the workspace and less memory can be used for the computational domain and the simulation results. Nowadays, memory of a high-end workstation or computing node of cluster is large. However, GRAM of a GPU is relatively small for the requirement of large-scale MHD simulations.

In this paper, we present a novel approach of handling the whole mesh (computational domain) block-by-block(we name it "block-based structure") to save to memory usage of GPU computing while retaining the performance using multi-GPU systems. Efficient adaptive mesh refinement (AMR) on multi-GPU systems is developed using our approach. An application to large-scale global magnetohydrodynamic (MHD) simulation of the solar wind interaction with Earth's magnetosphere is presented. Many optimization techniques are also introduced and huge amount of memory can be saved using AMR of our framework. As a result, we are able to enlarge the simulation domain to reproduce the full structure of the magnetosphere.

2. GPU Direct-MPI hybrid parallel data communication for distributed multi-GPU systems

Distributed multi-GPU systems or GPU clusters provide extremely high computation speed. However, the data transfer speed is comparatively slow. Hence, data communications between GPUs and computing nodes bottleneck the efficiency of numerical simulation using distributed multi-GPU systems. Enhancement of data communications directly reflect the total efficiency gain. A GPU Direct-MPI hybrid parallel data communication of our efficient CFD/MHD simulation framework for distributed multi-GPU systems has been presented in our previous work (1)(2). In this section, we would like to review our GPU Direct-MPI hybrid parallel communication in brief.

Instead of MPI, our GPU Direct-MPI hybrid parallel data communication utilizes GPU Direct 2.0 of CUDA (3) for intra-node data communication between the GPUs as shown in Fig. 1 which perform large speedup

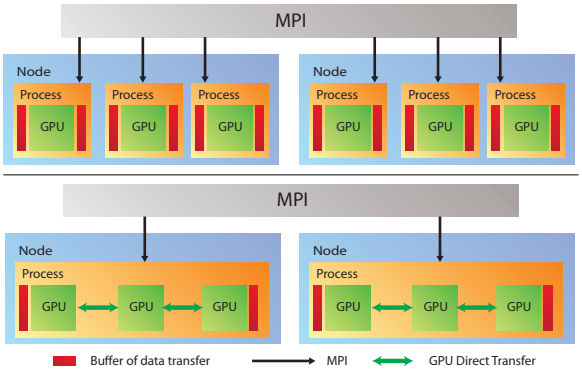

Fig. 1: Flat MPI (top) and our GPU Direct-MPI hybrid data communication (bottom)

for the data communication. On the other hand, the usage of host memory as the buffer for MPI data transfer can also be reduced. In our experiment tests, the total memory usage of an ideal MHD simulation with the resolution 1296^3 can be reduced to 81% (3D decomposition) and 60% (1D decomposition) running on 216 GPUs of TSUBAME 2.5 (4)(5).

Our GPU Direct-MPI hybrid parallel data communication is a 2-layer data communication model. Multiple GPUs within a single node is handled by a single process which is not related to MPI. On the top layer, users only have to consider that there is only one partition per node and there is only one MPI RANK for each partition (each node). On the GPU layer, the partition will be decomposed again and copy to each GPU. Moreover, data communication between GPUs is done by peer-to-peer data communication and no MPI is invoked in this layer. In 3D decomposition, the data communication in the other 2 directions are done by copying the data of each GPU from/to the different part of a single buffer. Boundary data to be copied to another node via MPI will be copied to a continuous memory space (device memory) for alignment and then being copied to a single host buffer. Non-blocking copy is used so that all the GPUs can copy the data from/to the host buffer simultaneously. As a result, Significant speedup is achieved by our GPU Direct-MPI hybrid parallel data communication.

3. A Block-Based Structure for Efficient AMR on Multi-GPU Systems

In this section, we describe the detail of our block-based structure. To explain the principle, we first take a look into to the numerical scheme of a partial differences equation (PDE) and see how it relates to a simulation program. As an example, a PDE is shown in Fig. 2, where the following descriptions are given:

- The physics quantities U and F are reflected to data storage and data structure (for example : meshes or particles)
- ∂x is related to the data access between the elements – the stencil computation.
- ∂t is related to the programming flow – the sequence of the simulation code

The definition of our block-structure for mesh-based simulation is given as the follows:

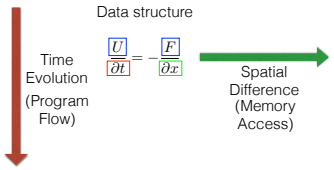

Fig. 2: The perspective of how a PDE relates to the components of a program

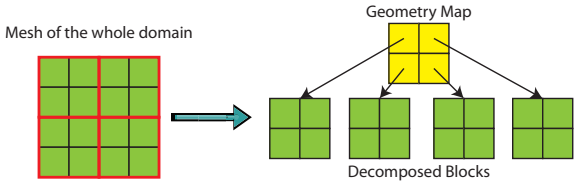

Fig. 3: The block-based structured mesh

- The whole mesh is decomposed into a fixed number of partitions (called blocks hereafter).
- Each block is a subset of the whole computational domain, where the resolution of each block may vary.
- Each block has only one neighbour in each direction $(x+, x-, y+, y-, z+, z-)$.
- A geometry map is used to link all the blocks, and also for finding the adjacent block (see Fig. 3).

Our block-based structure motives to give the flexibility in implementing different scheme/kind of simulation and efficiency in memory management. Multi-stream task-parallel and adaptive mesh refinement (AMR) based on this data structure will be introduced in the following sections.

3.1 Multi-stream Task Parallel on GPU

In general, the workflow of a mesh-based fluid simulation is :

(1) Calculate F^t from the governing equations using U^t (stencil computation)

(2) Evolve U^t to U^{t+dt} (for example, using the Runge-Kutta method)

(3) Boundary condition and addition processes such as limiters, smoothing, etc.

(4) Repeat from step 1 until the target time is reached.

In these processes, except the main physics quantities U, many other data such as F^t and the intermediate results of $t + \frac{dt}{n}$ of the Runge-Kutta method have to be generated. Each of these intermediate results generally requires the same number of data (grid points) of U for parallel computing. For example, even there is not a except resolution is mentioned, Fukazawa et al. reported that 64 MB/core had been used for the computational domain where additional 192 MB/core for the workspaces for the computation scheme in (6). In our block-based structure, a block is a unit to be processed. Therefore, the storage of the intermediate

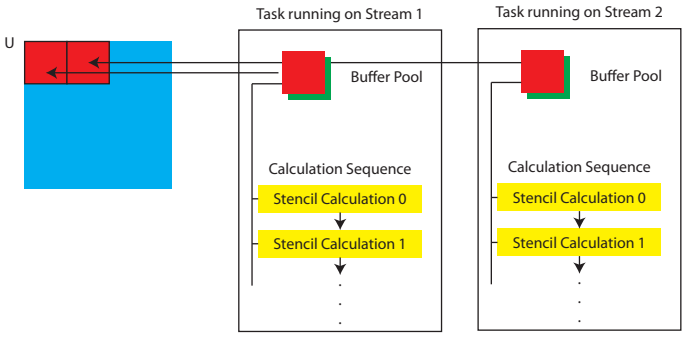

Fig. 4: Multi-stream task parallel on GPU with the block-based structure(same task)

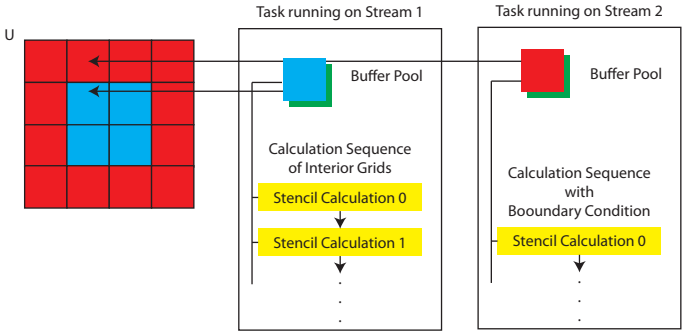

Fig. 5: Multi-stream task parallel on GPU with the block-based structure(different tasks)

results is only required to be the same size as the block. In fact, even on the many-core GPUs, it is not yet possible to process all the data of a large mesh at once. In addition, the memory space is defined as a buffer pool and will be reused to store the intermediate results in each calculation step. As a result, a lot of memory of the intermediate results is saved for a larger simulation domain. As shown in Fig. 4, the calculation sequence with the buffer pool is defined as a task, and multiple streams for concurrent kernel execution are applied to fully utilize the computing power of the GPU. Each stencil calculation is a CUDA kernel. Moreover, it is possible to launch multiple tasks with different calculation kernels. For example, as shown in Fig. 5, a task for the calculation of the interior grids and another task that calculates the boundary grid with the boundary condition are launched in parallel.

3.2 Adaptive mesh refinement

Adaptive mesh refinement techniques (AMR) that automatically adapt the computational grid to the solution of the governing PDEs can be very effective in treating problems with disparate length scales. Let the resolution of the mesh high enough only in regions deems of interest (for example the regions with high gradient), thereby saving orders of magnitude in computing resources for many problems. For example, in large-scale global MHD simulation of the solar wind interaction with magnetosphere, for typical solar wind flows, length scales can range from tens of kilometers in the near Earth region to the Earth-Sun distance (1 AU $\simeq 1.5 \times 10^{11}$m), and timescales can range from a few seconds near the Sun to the expansion time of the solar wind from the Sun to the Earth ($\sim 10^5 s$). The use of AMR is extremely beneficial for solving problems with

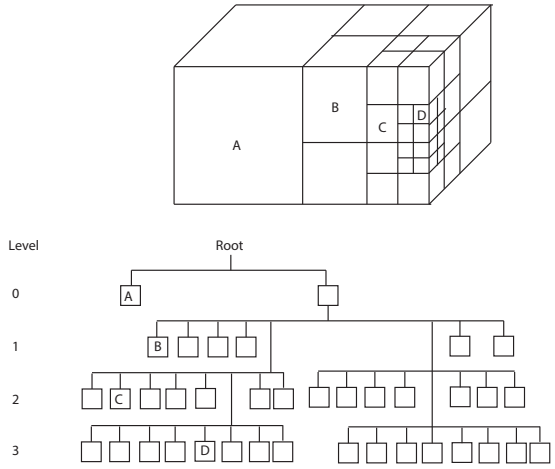

Fig. 6: Tree data structure for AMR

such disparate spatial and temporal scales.

In general implementations, a hierarchical tree data structure and additional interconnects between the "leaves" of the trees is used to keep track of mesh refinement and the connectivity between solution blocks. As shown in Fig. 6, the blocks of the initial mesh are the roots, which are stored in an indexed array data structure. Associated with each root is a separate "octree" data structure that contains all of the blocks making up the leaves of the tree which were created from the original parent blocks during mesh refinement. Each grid block corresponds to a node of the tree. Traversal of the tree structure by recursively visiting the parents and children of solution blocks can be used to determine block connectivity. However, in order to reduce overhead associated with accessing solution information from adjacent blocks, the neighbors of each block are computed and stored directly, providing interconnects between blocks in the hierarchical data structure that are neighbors in physical space.

AMR is a method that, partitions blocks of a mesh into different resolution (level) to save the computation time and memory usage. However, additional processes of calculations or data exchange of the halo between partitions at different levels are needed. Since data communication is relatively slow compare to computing, these additional process between grid blocks with different levels cause a huge overhead for AMR on multi-GPU systems. AMR generally uses tree data structure to mange the partitioning where each leaf has the same resolution. Therefore, grid blocks with different levels can have multiple neighbors in one direction. We found that this cause large overhead in data communication and greatly decreases the efficiency of AMR on GPUs. Many implementations of AMR on GPU such as (7) use the CPU to help data management and communication. However, such approaches cause a lot of data communications in copying the data of the leaves between GRAM and host memory in each calculation step. In our framework, all the data are stored in GRAM entirely until the simulation ends. AMR is straightforward to be iimplemented by changing the resolution of each block of our block-base structure (see Fig. 7). In our approach, number of neighbors in each direction is limited to 1 no matter the level between two neighboring blocks are different or not. This is very effective in reducing the overheads of data communication in AMR.

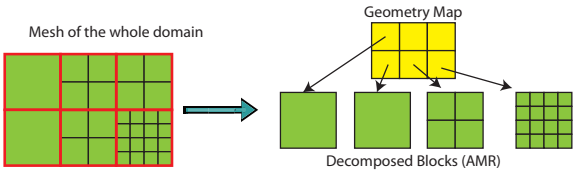

Fig. 7: AMR of the block-based structure

4. Application to Large-scale Global MHD Simulation

In this section, a large-scale global MHD simulation of solar wind interacting the Earth's magnetosphere implemented using our GPU Direct-MPI hybrid parallel framework will be presented. For applications to large-scale space simulation, we extend our global MHD simulation (8) for distributed multi-GPU systems using GPU Direct-MPI hybrid framework. Data communication between each GPU and computing node is handled by our framework. AMR is implemented using our block-based structure. In additional, several optimization techniques are introduced.

4.1 Simulation model and numerical scheme

Our global MHD simulations using multiple GPUs of a workstation can be referred to our previous work (8). It had been extend to large-scale using distributed multi-GPU system and presented in (1). The MHD equations are solved by a GPU implemented modified leapfrog scheme (see Fig. 8) for simulating the solar wind interacting with the Earths magnetosphere. The modified leapfrog scheme is used to solve the MHD equations and Maxwell's equations. The PDEs are listed as the follows :

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\mathbf{v}\rho) + D\nabla^2 \rho$$
$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v}\cdot\nabla)\mathbf{v} - \frac{1}{\rho}\nabla p + \frac{1}{\rho}(\mathbf{J}\times\mathbf{B}) + \mathbf{g} + \frac{\Phi}{\rho}$$
$$\frac{\partial p}{\partial t} = -(\mathbf{v}\cdot\nabla)p - \gamma p \nabla\cdot\mathbf{v} + D_p\nabla^2 p$$
$$\frac{\partial \mathbf{B}}{\partial t} = \nabla\times(\mathbf{v}\times\mathbf{B}) + \eta\nabla^2\mathbf{B}$$
$$\mathbf{J} = \nabla\times(\mathbf{B}-\mathbf{B}_d)$$

The modified leapfrog method is a combination of the two-step Lax-Wendroff scheme and the leapfrog scheme, proposed by Ogino et al.(9). In each sequence l, the first time step is calculated using the two-step Lax-Wendroff scheme and the subsequent $l-1$ time steps are calculated using the leapfrog scheme. These steps will be repeated until the calculation reaches the target time step. The modified leapfrog scheme adopts both the numerical stabilization of the two-step Lax-Wendroff scheme and the numerical attenuation of the leapfrog scheme to provide numerical results in a way that is much more stable than using one of the two schemes individually. The analysis and experiments in (9) suggest that $l = 8$ is the optimal number for the modified leapfrog scheme to provide stable results.

where ρ is the plasma density, $\mathbf{v}$ the velocity, $\mathbf{B}$ the magnetic field vector, and p the plasma pressure. $\mathbf{B}_d$ is the magnetic field of a planet. In our test case — the solar wind interaction with the Earth's magnetosphere, $\mathbf{B}_d$ is the magnetic dipole as the approximation of the Earth's magnetic field. $\Phi \equiv \mu\nabla^2\mathbf{v}$ is the viscosity. η is the resistivity, which was taken to be uniform throughout the simulation box with the range $0.0001 \le \eta \le 0.002$ for the magnetospheric configuration, $\mathbf{g} = -g_0/\xi^3$ ($\xi = \sqrt{x^2+y^2+z^2}$, $g_0 = 1.35\times10^{-7}$

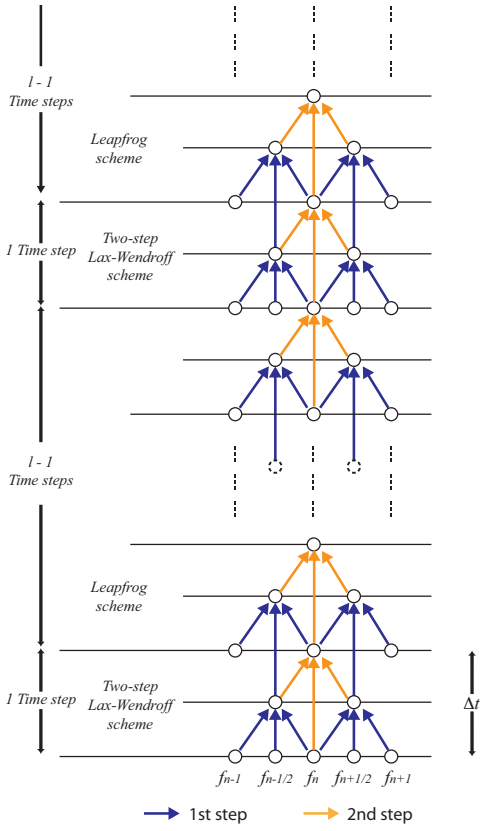

Fig. 8: Illustration of the modified leapfrog scheme

(9.8 m/s^2)) is the force of gravity, and $\gamma = 5/3$ is the ratio of specific heat. D is the diffusion coefficient of particles and D_p is the diffusion coefficient of pressure. Coefficient μ was artificially assigned in order to control numerical vibration of the short wavelength resulting from an initial value or a rapid magnetic field change, and let $D = D_p = \mu/\rho_{sw} = 0.001$, where ρ_{sw} is the solar wind density. $R_e = 6.37\times10^6$ m is the radius of the Earth. The simulation model is shown in Fig. 9. The interplanetary magnetic field (IMF) is set to $B_z = -5$ nT. Solar wind comes as the constant source along the x-axis from the upstream boundary at $x = x_0$ to the outflow boundary at $x = x_1$. For brevity, other detailed parameters and settings are spared here and can be referred to (1), (8) and (9).

4.2 Optimization techniques

In this section, several optimization techniques to enhance the performance of the simulation are going to be introduced. According to the definition, each block only has one neighbour in each direction which reduces the overheads of the data communication between grid blocks at different levels. However, there are still 9 neighbours (see Fig. 10) between one pair of the high and low level grids in a three dimensional simulation and the overhead is quite big. We found that when updating the halo region of the low level grid points from the high level grid points (we refer to this as H2L copy in the following sections. Conversely, updating the high level grid points from the low level grid points will be referred to as L2H copy.) using linear interpolation, halo grid points of the high level grid can be ignored. On the other hand, if the halo region of the low level grid is updated, the halo grid at the corner as well as the border of the high level grid can also be updated from the ad-

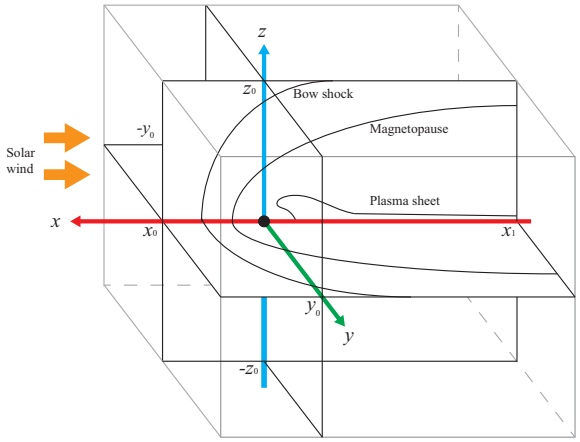

Fig. 9: Simulation domain of solar wind interacting with the Earth's magnetosphere

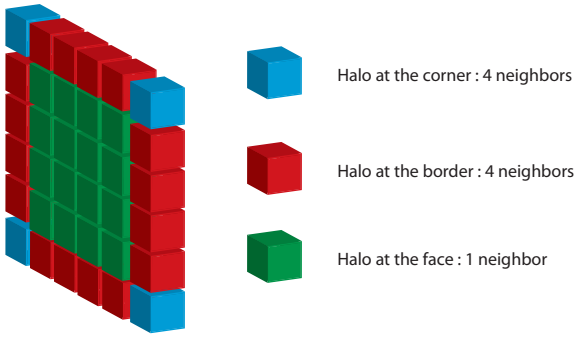

Fig. 10: The 9 neighbours for updating one slice of the halo (in one direction) of a 3D mesh-based simulation.

jacent grid (as shown in Fig. 11). Therefore, to achieve further speedup, the following strategy is applied:

(1) Copy the data between all blocks with the same level.

(2) H2L copy with 9 neighbors in each direction (ignores the duplicated corner and border).

(3) L2H copy with 1 neighbor in each direction.

In this data communication sequence, neighbor access in the L2H copy is reduced to 1. Besides, peer-to-peer access of GPU Direct is used in the interpolation kernel to update the halo grid points. Therefore, no additional buffer for storing the interpolation results is required, resulting in savings in memory usage.

In AMR, refine the resolution (changing the level) of a block is time consuming, especially for GPU computing. According to the modified leapfrog scheme (Fig. 8), when the Lax-Wendroff scheme is processed every 8 steps, value of the of grid points of $(i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2})$ will be interpolated again. The results of last step does not have to be kept. Therefore, we only process the refinement every 8 steps to skip the interpolation of the grid points of $(i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2})$ itself.

Constant value such as the dipole field $\mathbf{B}_d$ which is invoked in the simulation, is stored in texture memory. Texture memory is a special feature of GPU which not only provides fast read with a specific texture cache, but also perform fast linear interpolation. Even though it

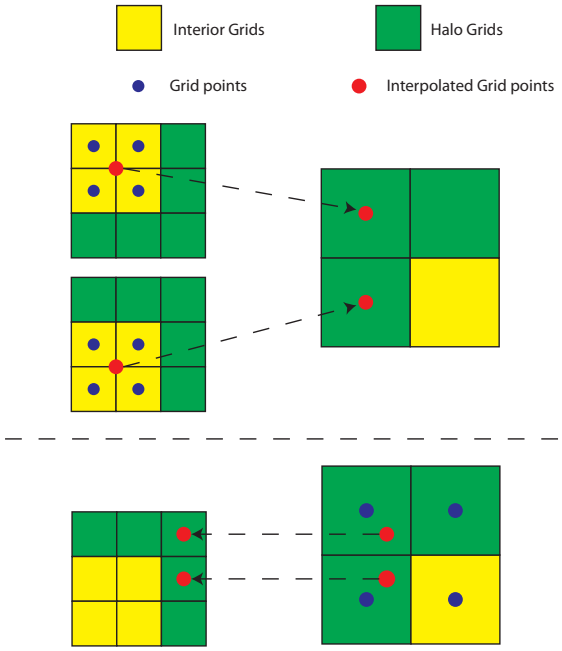

Fig. 11: The interpolation of grid points between different level

only supports single precision, the accuracy is enough for the constant value.

5. Results and analysis

Performance tests of large-scale global MHD simulation achieved 4.38 TFLOPS in double precision for a computational domain of $1980 \times 1320 \times 1320$ using 216 GPUs using our efficient GPU Direct-MPI hybrid data communication. The results had been shown in our previous work (1). In this paper, we would like more focus on the new block-based structure as well as our AMR simulation implemented using our framework. First, to show the efficiency of our block-based structure on GPU computing, measurement of our global MHD simulation on a single GPU will be discussed. In our tests without AMR, memory usage of our MHD simulation with a resolution of 256^3 requires 1.025 GB for the main physics quantities, another 1.025 GB for the physics quantities at the half grid points (needed by the modified leapfrog scheme) and 2.45 GB for the workspace. By using our block-based structure and decomposed the mesh into $4 \times 4 \times 4 = 64$ blocks, the memory usage is 1.125 GB for the main physics quantities, another 1.125 GB for the physics quantities at the half grid points and 0.056 GB for the workspace, respectively. A huge number of memory of 2.394 GB is saved for the workspace. Even though 0.1 GB is increased in the usage of main physics quantities and the half grid points data, because of the buffer (halo) for the data communication between the blocks. In total, our block-based structure save 2.194 GB.

In application our framework to global MHD simulation, the gradient of the pressure field is used as the metric to determine the blocks with the highest resolution when applying AMR to the simulation. In our experiment, AMR in three dimension largely increase the overhead of data communication between the blocks with different level. On the other hand, additional process for load balancing between each GPU is needed. Therefore, we only refine the mesh along the x-direction. As it shown in Fig. 12, the blocks of the large gradient of

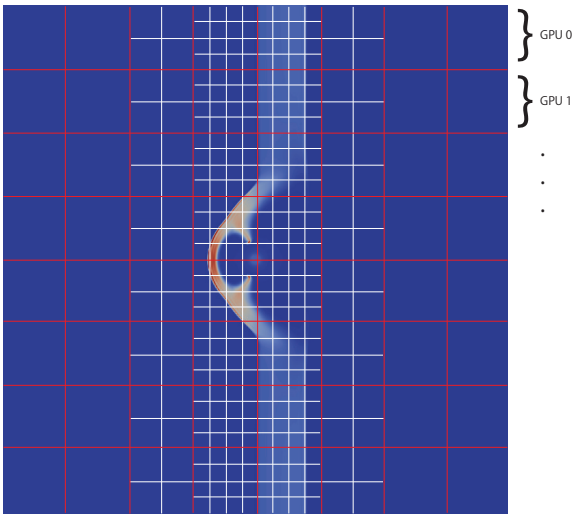

Fig. 12: Image illustrates the blocks with different level (Note that it is not the real resolution). Refining the mesh along x-direction and distribute to each GPUs by the other direction for less communication overhead and well load balancing. Red lines show the partition of the blocks.

the pressure field are refined to the highest level. The neighboring blocks in y-direction(similarly, z-direction too) are also refined as the same level. In this method, H2L and L2H copy only needed in x-direction. Moreover, we distribute the partition domains to each GPU in y and z direction. So that all the GPUs contain the same number of grid points (load balance). In our performance test with 512^3 resolution using 2 K40c and 2 K80 GPUs. Our approach reduce the total memory usage from 4.670 GB to 3.504 GB for each GPU and speedup the simulation from 1070.19 ms/step to 938.406 ms/step. Therefore, 25% memory save and 13% speedup are achieved. However, the process of refining the blocks requires additional 7304.87 ms to process the refinement even though using our optimization technique to reduce the interpolation. How frequent the mesh is being refined is depending on the testing problem, the user defined height level of spatial scale and the timescale δt. In our tests, the frequency of mesh refinement is varying but never happens within every 128 steps. If we add up a approximate refinement $(7304.87/128) = 57.069$ ms/step to the total elapse time, which is $938.406 + 57.069 = 995.475$ ms/step, 7% speedup is achieved in total efficiency.

Use of the block-base structure allows us to save the memory usage.Therefore, we able to extend the simulation domain from $(x, y, z) = (-60R_e, -30R_e, -30R_e) \sim (30R_e, 30R_e, 30R_e)$ to $(x, y, z) = (-70R_e, -70R_e, -70R_e) \sim (70R_e, 70R_e, 70R_e)$ with the same multi-GPU system. Therefore, the simulation domain is enlarged about 8.47× (as shown in Fig. 13). The extended domain is large enough to cover the boundary of the bow shock as well as the whole magnetosphere in the y and z direction.

It is not only extend the simulation domain for investigating the natural phenomenon in the widely region of the space. But also benefit in improving the simulation model. Due to the lack of computational resourcese, the simulation domain of the solar wind and Earth's

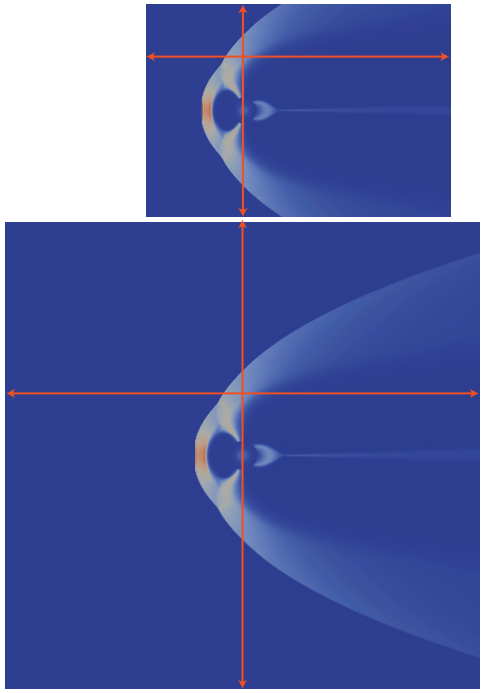

Fig. 13: Simulation domain enlarge from $(x, y, z) = (-60R_e, -30R_e, -30R_e) \sim (30R_e, 30R_e, 30R_e)$ to $(x, y, z) = (-70R_e, -70R_e, -70R_e) \sim (70R_e, 70R_e, 70R_e)$ (8.47×) using AMR implemented with our framework

magnetosphere interaction introduced many existing researches including the model introduced by Ogino et al. 9 doesn't include the whole magnetosphere in y and z direction. Neumann boundary condition at 45 degrees to the x-axis was applied to to $y = y_0$ and $z = z_0$ boundary as shown in the following Fig. 14. The reason for this boundary condition setting is the assumption of making the magnetic field align with the blow shock but the computation is not efficient and gives some restrictions in decompose the domain because of the needed of accessing the neighboring grid points along 45 degree. This problem is solved and the Neumann boundary condition (Fig. 15) is applied to the y and z boundary in our extended simulation domain using AMR. Resulting in simpler and efficient calculation of the y and z boundary condition.

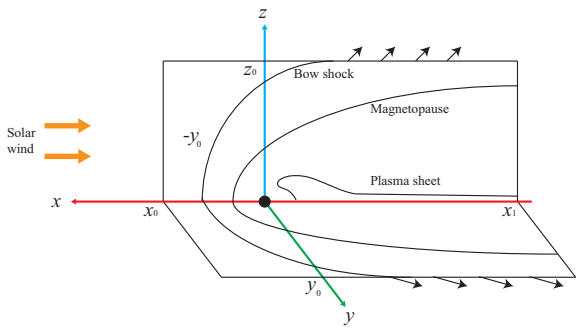

Fig. 14: The boundary condition at 45 degrees to the x-axis

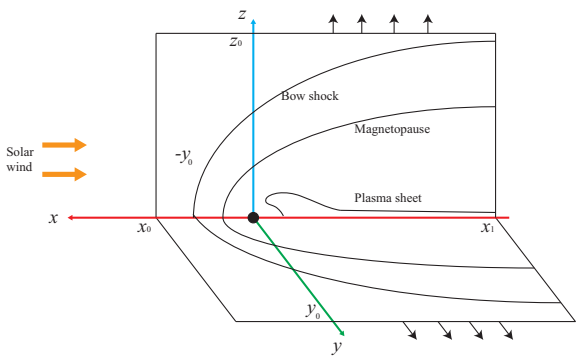

Fig. 15: Enlarged simulation domain which contains the whole magnetosphere with Neumann boundary condition.

3D Visualization of the simulation results (p:pressure) of our global MHD simulation of the whole magnetosphere with AMR are shown in following Fig.16 and Fig.17.

6. Conclusion

Performing large-scale MHD simulations on distributed multi-GPU systems provides us an opportunity to perform numerical simulations in a more efficient manner. However, the overhead caused by the data communication bottlenecks simulation performance. To overcome this problem, we proposed a GPU Direct - MPI hybrid framework and achieved significant enhancement compared to flat MPI implementations of fluid simulations using multi-GPU systems. GPUs provide extremely high computing power but the amount of memory is relatively small. Adaptive Mesh Refinement is a effective technique to reduce the memory usage and computing time for large-scale simulation. However, the memory access and data management of GPU is very slow compare to its computing power. This cause the difficulty in implementing an efficient AMR on GPU. In many cases, traditional implementation method of AMR on CPU lost the benefit when applying to GPU.

In this paper, a new block-based structure for task parallel and efficient adaptive mesh refinement on multi-GPU systems is developed to save the memory usage while retaining the efficiency. Application of our framework to global MHD simulation as well as several techniques are introduced. Simulation results and analysis are shown. In our global MHD simulation of the solar wind interaction with the Earth's magnetosphere, we use low level blocks in the region of in front of the bow shock to achieve the same resolution of the magnetosphere as a 512^3 mesh using 2 K40c and 2 K80 GPUs. Our AMR reduces the 25% memory usage and about 7% speedup the simulation is achieved. By fully utilized the memory of the GPUs using AMR of our framework, we are able to enlarge the simulation domain to cover a widely region of the space which contains the whole magnetosphere. Inter-node data communication via MPI is still a bottleneck since a D2H and H2D copy is needed before and after the MPI communication. GPU Direct RDMA is a feature that can speedup the inter-node data communication. Unfortunately it is not yet available on TSUBAME 2.5. We are looking forward to improve the efficiency using GPU Direct RDMA after the next upgrade of TSUBAME completes.

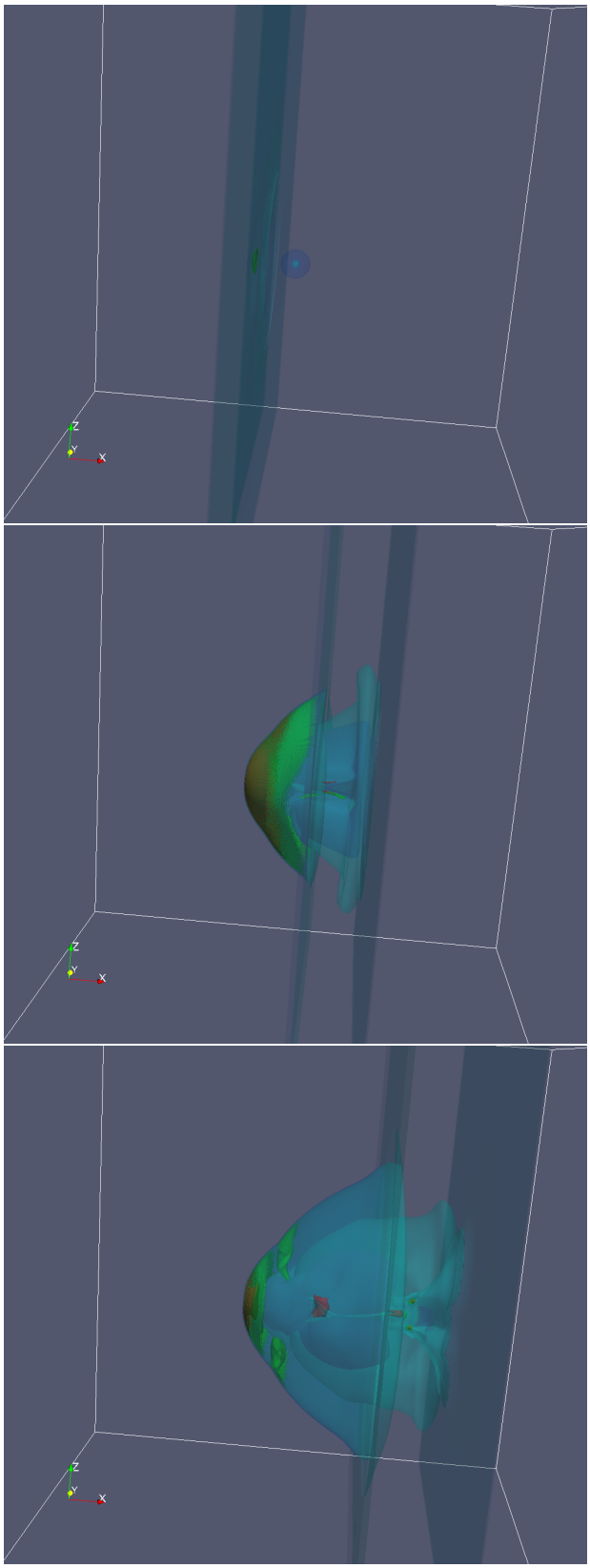

Fig. 16: 3D visualization of our simulation results (p) of the whole magnetosphere interacting with the solar wind

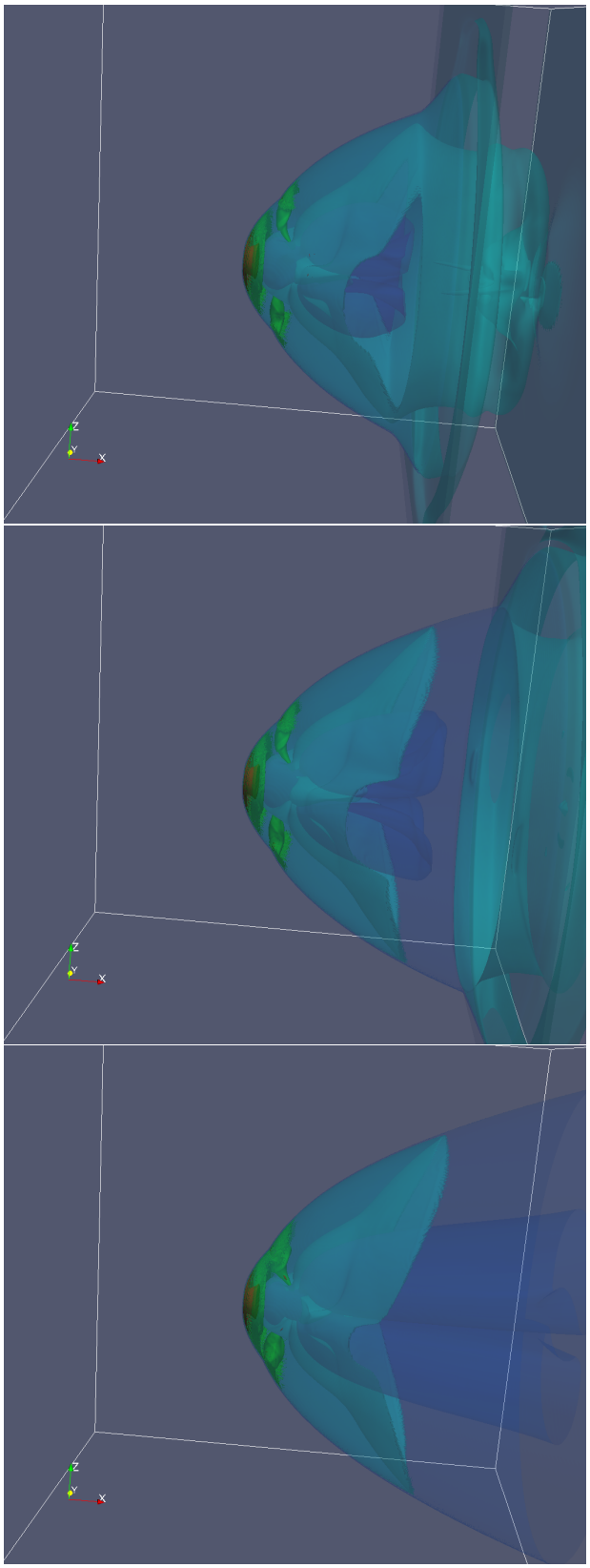

Fig. 17: 3D visualization (p)of our simulation results (p) of the whole magnetosphere interacting with the solar wind (cont.)

Acknowledgements

This research was supported in part by the Japan Society for the Promotion of Science (KAKENHI), Grant-in-Aid for Scientific Research (B) 23360046 and Grant-in-Aid for Scientific Research (S) 26220002 from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan, and Japan Science and Technology Agency (JST) Core Research of Evolutional Science and Technology (CREST) research programs on Highly Productive, and High Performance Application Frameworks for Post Petascale Computing.

References

(1) U.H. Wong, T. Aoki, and H.C. Wong, "Efficient large scale MHD simulations: 3D OT vortex and Solar Wind-Earth interaction using distributed multi-GPU system", Proc. 27th CFD Symp., (2013).

(2) U.H. Wong, T. Aoki, and H.C. Wong, "Efficient magnetohydrodynamic simulations on distributed multi-GPU systems using a novel GPU Direct-MPI hybrid approach", CPC, 185(2014), pp. 1901-1913.

(3) `https://developer.nvidia.com/category/zone/cuda-zone`

(4) T. Endo, A. Nukada, S. Matsuoka, and N. Maruyama, "Linpack evaluation on a supercomputer with heterogeneous accelerators", Proc. 24th IEEE International Parallel and Distributed Processing Symp. (IPDPS10), 2010.

(5) G. S. Information, T. I. o. T. Computing Center, TSUBAME 2.5 hardware software specifications, `http://www.gsic.titech.ac.jp/sites/default/files/spec25e1.pdf`

(6) K. Fukazawa and T. Umeda, "Performance measurement of magnetohydrodynamic code for space plasma on the typical scalar type supercomputer systems with the large number of cores", International Journal of HPC Applications, (2012).

(7) H.Y. Schive, Y.C. Tsai, and T. Chiueh, "GAMER: A graphic processing unit accelerated adaptive-mesh-refinement code for strophysics", The Astrophysical Journal Supplement Series, 186(2)(2010), pp. 457-484.

(8) U.H. Wong, H.C. Wong, Y. Ma, "Global magnetohydrodynamic simulations on multiple GPUs", CPC, 185(2014), pp. 144-152.

(9) T. Ogino, R. J. Walker, and M. Ashour-Abdalla, "A global magnetohydrodynamic simulation of the magnetosheath and magnetopause when the interplanetary magnetic field is northward", IEEE Trans. on Plasma Sci., 20(1992), pp. 817-828.

(10) T. Shimokawabe, T. Aoki, and N. Onodera, "Framework for Block-type AMR method on GPU computing", HPCS2013(2013) pp. 156-165.