論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Large-scale Global MHD Simulation for Solar Wind Interaction with Magnetosphere on Multi-GPU Systems
著者(和文)	WONGUnhong
Author(English)	Un Hong Wong
出典(和文)	学位:博士（工学）, 学位授与機関:東京工業大学, 報告番号:甲第9998号, 授与年月日:2015年9月25日, 学位の種別:課程博士, 審査員:青木　尊之,長崎　孝夫,肖　鋒,長谷川　純,横田　理央
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第9998号, Conferred date:2015/9/25, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Tokyo Institute of Technology

Doctoral Thesis

Large – scale Global MHD Simulation for Solar Wind Interaction with Magnetosphere on Multi – GPU Systems

Author:
Un Hong Wong

Supervisor:
Prof. Aoki Takayuki

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Engineering

in the

Interdisciplinary Graduate School of Science and Engineering
Department of Energy Science

August 2015

TOKYO INSTITUTE OF TECHNOLOGY

Abstract

Interdisciplinary Graduate School of Science and Engineering

Department of Energy Science

Doctor of Engineering

Large – scale Global MHD Simulation for Solar Wind Interaction with Magnetosphere on Multi – GPU Systems

by Un Hong Wong

Nowadays, the physics of the space environment is not only a frontier topic in science but is also closed to our daily live. Understanding the environment in space is becoming increasingly important. To investigate large-scale space phenomenon such as the solar wind interacting with the planet's magnetospheres, global magnetohydrodynamic (MHD) simulation with huge amount of computations is often used. The recent development of modern graphics processing units (GPUs) makes the simulation process be performed in a more efficient manner. GPUs have been widely utilized in magnetohydrodynamic (MHD) simulations in recent years. However, due to the limited memory of a single GPU, the use of multi-GPU systems needs to be explored for large-scale MHD simulations.

In this dissertation, we firstly clarify the limitation of executing the current space simulation caused by the computational resources, which then motivates us to develop a large-scale global MHD simulation code that efficiently runs on multi-GPU systems. A GPU Direct-MPI hybrid framework for efficient data communication of large-scale grid-based simulation on multi-GPU systems is presented. Large-scale global MHD simulation of the solar wind interaction with Earth's magnetosphere with in-situ visualization has been implemented. The magnetosheath and polar cusp caused by the solar wind interacting with the Earth's magnetosphere are examined. A new block-based structured for efficient Adaptive Mesh Refinement (AMR) is also presented. With AMR we improve the simulation model and enlarge the simulation domain to reproduce the whole structure of the bow shock. The variation of the size of the magnetosphere is found. Moreover, the variation affect to the period of the Moon staying in different part of the magnetosphere, which is very important in investigating the evolution of the Moon. Application of our simulation to solar wind-Earth's magnetosphere interaction for the geomagnetic reversal is also carried out.

Acknowledgements

At first, I would like to spend my deepest gratitude to Professor Aoki, my supervisor, for accepting me as one of his Ph.D. students. His solid foundation and hard working give me a good example of a researcher, and is always tolerant to me and cheer me up when my daze disappointed him. My great thankful and apologize to my mother for allow his stubborn son to leave her alone and chase his own dream. My appreciation to all the members of Aoki Lab for sharing laughs and tears together. My deeply thanks to Professor Zesheng Tang and Dr. Hon-Cheng Wong of Macau University of Science and Technology. Without meeting them, I would give up to go to my master course. Special thanks to Dr. Xiao and Dr. Nagasaki of Department of Energy Sciences, Professor Nagai and Professor Tsunakawa of Department of Earth and Planetary Sciences for teaching me and treating me as a regular student when I audited their courses. My sincere gratitude to the Ministry of Education Culture, Sports, Science and Technology (MEXT) Japan for giving me the scholarship to support my study in Japan for the past three and half years.

Contents

List of Figures

List of Tables

Abbreviations

ALU	**A**rithmetic **L**ogic **U**nit
DMA	**D**irect **M**emory **A**ccess
D2D	**D**evice **To D**evice
D2H	**D**evice **To H**ost
FLOPS	**F**loating-point **O**perations **P**er **S**econd)
GPU(s)	**G**raphics **P**rocessing **U**nit(s)
GPGPU	**G**eneral-**P**urpose computing on
H2D	**H**ost **To D**evice
HPC	**H**igh **P**erformance **C**omputing
IMF	**I**nterplanetary **M**agnetic **F**ield
MHD	**M**agneto-**H**ydro-**D**ynamic
MPI	**M**essage **P**assing **I**nterface
MUSCL	**M**onotonic **U**pstream-**C**entered **S**cheme for **C**onservation **L**aws
OpenCL	**Open C**omputing **L**angurage)
OpenGL	**Open G**raphics **L**angurage)
PDEs	**P**artial **D**ifferential **E**quations
PIC	**P**article-**I**n-**C**ell
TVD	**T**otal **V**ariation **D**iminishing

Symbols

$\mathbf{B}$	magnetic field
$\mathbf{B}_d$	magnetic dipole field
$\mathbf{B}_{IMF}$	interplanetary magnetic field
D	diffusion coefficient of particles
D_p	diffusion coefficient of pressure
E	energy
$\mathbf{E}$	electric field
$\mathbf{g}$	gravity
g_0	standard Gravity
$\mathbf{J}$	current
k_B	Boltzmann constant
m	mass
n	number density
$\mathbf{p}, p$	pressure
P^*	total pressure
p_{sw}	pressure of the Solar Wind
q, e	charge
r	Euclidean distance
R_e	radius of Earth
T	thermal (temperature)
T_{sw}	thermal of the Solar Wind
$\mathbf{v}$	velocity vector
v_{sw}	velocity of the Solar Wind
ρ	density
ε_0	dielectric constant

η	resistivity
γ	heat capacity ratio
μ_0	magnetic permeability

In memory of my father

Chapter 1

Introduction

1.1 General Introduction

Human have been interested in the starry night sky since ancient history. For thousands of years, study of the stars/space — astronomy and related subjects — was conducted by the naked eye. Even from these early observations, many notable achievements were made such as the armillary sphere, which models the sky as well as the orbit of the astronomical objects. After the advent of the telescope in the 17^{th} century, followed by the very first Luna program and Apollo project, the human race has entered the era of space exploration. In recent years, many countries including Japan, China, and India also focus on developing their own space exploration projects. For example, the lunar missions SELENE project (Japan), Chang'E project (China) and Chandrayaan project (India). The United States also conducted the lunar exploration LRO (Lunar Reconnaissance Orbiter) project and had claimed that Mars will be the main target in the following years, while in the past they had been focusing on the artificial orbiting construction — the International Space Station (ISS) for many years. The development of the space technology eventually improves the convenience and the quality of our life. For example, artificial satellites play an important role in many aspects in our daily life, such as the communication network, weather observation, space-based satellite navigation system etc..

The theory of solar wind and the plasma environment of space had been suggested in the 19^{th} century. After the first measurement of the strength of solar wind by the Soviet satellite Luna 1 in 1959, it was proved that space is not vacuum. The stream of plasma – the solar wind flows outward from the upper atmosphere of the Sun. It consists of mostly electrons, protons and alpha particles with energies usually between 1.5 and 10 keV. The solar wind fills the region which covers the solar system. It is very important

to understand the plasma environment of the space - the space weather in applications and explorations to the space. Especially the electromagnetic induction caused by the plasma shock can damage electronic devices used in artificial orbiting objects. On the other hand, launching a rocket to carry the orbiting object escaping from the Earth's gravity requires a lot of fuel. The orbiter/spacecraft also needs fuel for thrusting in space. Fuel carried within the orbiter is limited and can not be refill after it has been launched to the space. Therefore, the use of the electron/ion, photon released from the sun as well as the magnetic field of the planet is used to produce thrust to the orbiter/spacecraft. As one of the recent achievements, the world's first interplanetary solar sail spacecraft "IKAROS" (Interplanetary Kite-craft Accelerated Radiation of the Sun) had been successfully launched to Venus by JAXA (Japan Aerospace Exploration Agency) on 21^{st} May 2010.

Space weather is a branch of space physics and aeronomy concerned with the time varying conditions within the Solar System, including the solar wind, conditions in the magnetosphere, ionosphere and thermosphere. The interest in space weather expanded in many businesses since space weather phenomena can interfere with or damage the communications satellites, weather satellites and the satellites of the Global Positioning System (GPS). Many countries have their own space weather prediction/forecast/research center to provide real-time observation, analysis and research to space weather. Beside, the U.S. National Space Weather Program [8] had been carried out by several U.S. government agencies in 1994 motivated to speed improvement of space weather services. Similar to the weather of the atmosphere, except some big events such as solar storms, small changes occur frequently in the space. Figure 1.1 shows the observation data of the solar wind from the the NASA (National Aeronautics and Space Administration) Advanced Composition Explorer (ACE) satellite [1] in 2 weeks (From 4^{th} July to 17^{th} July). For example, the velocity of the solar wind increased from 300 km/s to 600 km/s in one day (4^{th} July to 5^{th} July).

Space is vast. Observations depend on orbiters or telescopes that have high cost, and the observation region is limited. Therefore, numerical simulation has become an important tool to help understand the space environment. The development of scientific computing and the term "numerical experiment" manifest the usability, reliability and importance of numerical simulation in various modern science disciplines. Global magnetohydrodynamic (MHD) models play a major role in investigating large-scale space phenomenon such as solar wind — magnetosphere interaction. But the huge computation requirement in global MHD simulations is also a problem that needs to be solved. Most of the existing simulations to the magnetosphere only included the part of the magnetosphere, because the computational domain is so huge to cover the whole magnetosphere. Thus,

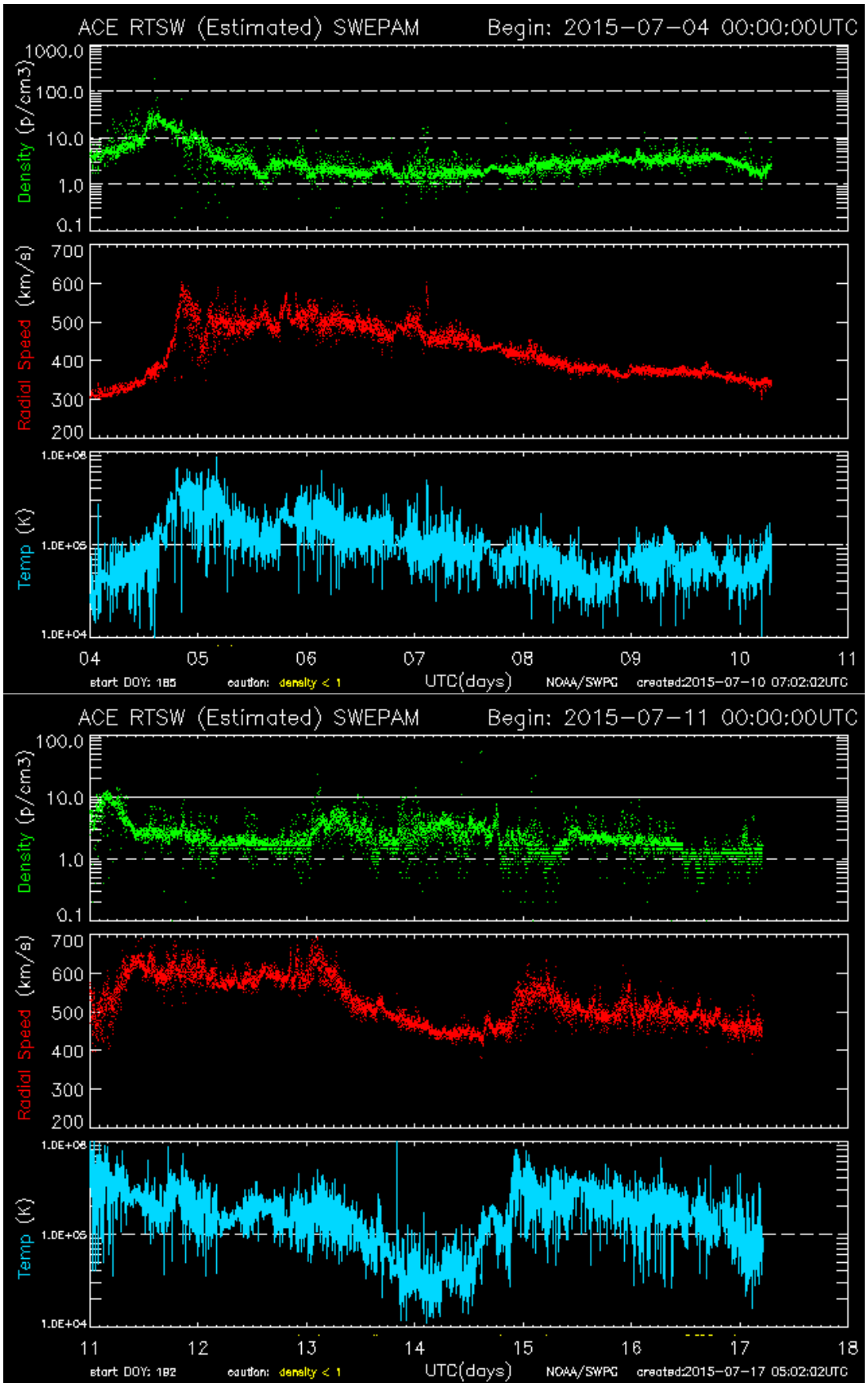

FIGURE 1.1: Solar wind observation data from the The NASA Advanced Composition Explorer (ACE) satellite [1].

the variations of the whole magnetosphere under different situation of the solar wind are important but had not yet be examined.

Graphics processing units (GPUs) have been widely utilized in MHD simulations in recent years. Due to the limited memory of a single GPU, multi-GPU systems are needed for large-scale MHD simulations. However, the data transfer between GPUs limits the efficiency of the simulations on such systems.

In this thesis, we study the limitation of the current space simulation caused by the computational resource. We present an efficient large – scale global MHD simulation of the solar wind interacting with the planet's magnetosphere using multi-GPU systems, including the GPU-rich supercomputer TSUBAME 2.5 of Tokyo Institute of Technology. Our approach not only increases the efficiency but also extends the usability of the global MHD simulation. The full structure of the Earth's magnetosphere is examined. The variation of the magnetosphere under different velocities of the solar wind are simulated and investigated. In our experiment results, the size of the structures of the magnetosphere including the bow shock as well as the magnetosheath is vary in the vertical direction (pole direction), but no obviously changes had been found in the horizontal direction (equator plane).

This thesis organized as follows: Chapter 1 is a brief introduction to space plasma and MHD simulation, the research background and motivation, as well as GPGPU and GPU systems. Related work of the state of the art GPU accelerated plasma simulation and our single GPU implemented ideal MHD simulation described in Chapter 2. Chapter 3 presents our GPU Direct-MPI hybrid data communication framework for efficient large-scale MHD simulations on distributed multi-GPU systems. Performance measurements of running our ideal MHD simulation of the GPU-rich TSUBAME supercomputer are given to show the efficiency. Implementation of large-scale global MHD simulation and practical applications to solar wind interaction with the Earth's magnetosphere are given in Chapter 4. Application of our simulation to the geomagnetic reversal is also carried out. In Chapter 5, the novel block-based structure for efficient AMR on multi-GPU systems and simulations of the whole bow shock are presents. And we conclude our work with explain the expectations of the possible future work in Chapter 6.

1.2 Magnetosphere

A magnetosphere is the area of space near an astronomical object in which charged particles are deflecting by that object's magnetic field. The structure of a magnetosphere is illustrated in Figure 1.2. Near the surface of the object, the magnetic field lines

resemble those of a magnetic dipole. Farther away from the planet surface, the magnetic field lines are significantly distorted by electric currents of the plasma flow. When talking about the Earth, magnetosphere is typically used to refer to the outer layer of the ionosphere.

The solar wind is kept away by the pressure of the Earth's magnetic field. The magnetopause, the area where the planetary magnetic field is balanced with the pressure from the solar wind, is the boundary of the magnetosphere. Although the Earth's magnetic field is approximately a symmetric dipole field, the magnetosphere is asymmetric. The frontside (sunward side) of the magnetosphere is being compressed to about 10 R_e (radius of the Earth) out but the tail side stretching out in that being extended beyond far beyond (about 200 R_e) by its interaction with the solar wind.

The tail side of the magnetosphere is called the magnetotail. It contains two lobes, referred to as the northern and southern tail lobes. The magnetic lines of the northern tail lobe points towards the object and the southern tail lobe points away. The tail lobes are almost empty, with few charged particles opposing the flow of the solar wind. The two lobes are separated by a plasma sheet, an area where the magnetic field is weaker and the density of charged particles is higher.

The inner region of the Earth's magnetosphere is the plasmasphere. It is located above the ionosphere and is consisting of low energy plasma (cool plasma). The outermost layer of frontside of the magnetosphere is the bow shock, where the solar wind slows abruptly. The polar cusp is a region in which the magnetosheath plasma has direct access to the ionosphere. The magnetosheath is the region of the magnetosphere between the bow shock and the magnetopause. It is formed mainly from shocked solar wind, though it contains a small amount of plasma. The regularly magnetic field generated by the planet becomes weak and irregular in the magnetosheath due to interaction with the incoming solar wind, and is incapable of fully deflecting the highly charged particles. This is caused by the collection of solar wind gas that has effectively undergone thermalization. It acts as a cushion that transmits the pressure from the flow of the solar wind and the barrier of the magnetic field from the object. The Earth's magnetosheath typically occupies the region of the space approximately 10 R_e on the frontside of the Earth, where on the tail side it is extended farther out due to the pressure of the solar wind. The exact location and width of the magnetosheath does depend on variables of the solar wind. The motivation of our simulation is to examine the variations of the magnetosphere's structure including the magnetosheath under different situations.

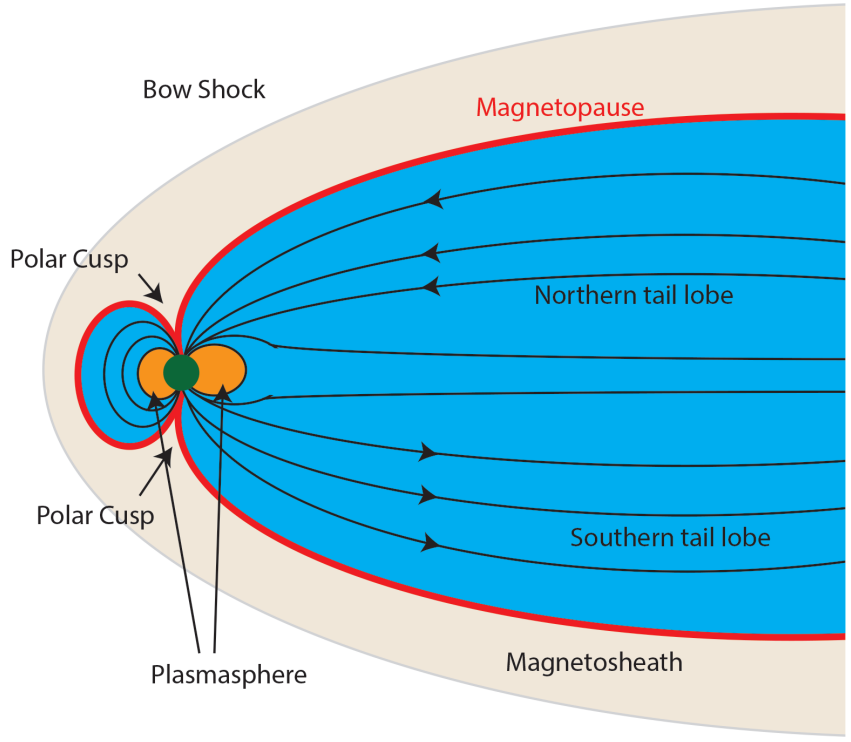

FIGURE 1.2: Schematic of the structure of Earth's magnetosphere.

1.3 Space Plasmas and MHD Simulations

Space plasma is the plasma that occurs naturally in the universe. Description of the space plasma spans the scale from single particle (micro): an element with a distribution function in 6D phase space(position and velocities) based on the plasma kinetic theory, to a bulk of electrically conducting fluid (macro, the fluid description). The simulation form micro scale to macro scale is listed as following :

1. Full particle model
2. Particle-in-cell — Charged particles interact with magnetic fields of (generally the cell face) a mesh
3. Hybrid model — Heavy ions as particles where electrons behave as massless fluid
4. Two-fluid — Ions and electrons
5. One-fluid — The magnetohydrodynamics (MHD), only heavy ions are considered since electrons are 1836× lighter.

Since density of space plasma is very low (several particles cm^{-3}) and the temperature is very high (order of $10^6 \sim 10^8$), its mean-free path becomes very long. For an example, the mean-free path of solar plasma is about 1 AU (Astrophysical Unit = the distance from the Sun to the Earth). Thus, space plasma is assumed to be collisionless. In collisionless plasmas, the direct binary collisions between two particles are so infrequent compared with any relevant variation in the fields or the time scales of particle dynamics that they can be safely neglected. In such plasmas the mean free path of a particle is much larger than the typical scales of interest (The Knudsen number $Kn \gg 1$). Most plasmas in space are sufficiently dilute to belong to this type. Therefore, particles interact with each other not through Coulomb collisions, but by the emission and absorption of collective excitations of the plasma (long-range Coulomb interaction). Each charged particle "feels" the long-range Coulomb force with a bulk of charge particles surrounding it. In plasma, the potential produced by an external point charge is

$$\Phi(r) = \frac{q}{4\pi\varepsilon_0 r} \exp(-\frac{r}{\lambda_D}) \tag{1.1}$$

where r is the distance to the point charge and ε_0 is the dielectric constant. λ_D is the Debye length — the measure of a charge carrier's net electrostatic effect in solution, and how far those electrostatic effects persist. It is given by

$$\lambda_D = \sqrt{\frac{k_B T}{4\pi n e^2}} \approx 6.9\sqrt{\frac{T}{n}} \tag{1.2}$$

where k_B is the Boltzmann constant, n the number density (uint : cm^{-3}), T the temperature, and e the charge. It can be found that in the above Equation 1.1, the Coulomb potential is exponentially reduce with r/λ_D. There is a influence when the distance of the charge carrier is $r \sim \lambda_D$. A volume(sphere) whose radius is λ_D, is called a Debye sphere. Charges outside the Debye sphere are electrically screened. In space plasmas where the electron density is relatively low, the Debye length may reach macroscopic values, such as in the solar wind and magnetosphere. For example, taking the typical values of temperature and density of the solar wind ($n = 5$ cm^{-3}, $T : 5 \times 10^4$ K), the Debye length of the space plasma is 690 cm. The kinetics of collisionless plasma is described by the Vlasov equation:

$$\frac{\partial f_s}{\partial t} + \frac{\partial f_s}{\partial \mathbf{r}} + \frac{q_s}{m_s}(\mathbf{E} + \mathbf{v} + \mathbf{B})\frac{\partial f_s}{\partial \mathbf{v}} = 0 \tag{1.3}$$

where $\mathbf{E}$, $\mathbf{B}$, $\mathbf{r}$ and $\mathbf{v}$ represent the electric field, the magnetic field, the position, and the velocity, respectively. The subscript s represent to the species of the particles (for example, $s = i$ for ions and $s = e$ for electrons). $f(\mathbf{r}, \mathbf{v}, t)$ is the distribution function describes the number of particles of the species s in a position-velocity phase space.

The quantities q_s and m_s are the charge and mass of singly-charged particle. Spatial-temporal development of electromagnetic fields is described by the full-set of Maxwell's equations (Vlasov-Maxwell system):

$$\begin{aligned} \boldsymbol{\nabla} \times \mathbf{B} &= \mu_0 \mathbf{J} + \frac{1}{c^2} \frac{\mathbf{E}}{\partial t} \\ \boldsymbol{\nabla} \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\ \boldsymbol{\nabla} \cdot \mathbf{E} &= \frac{\rho}{\varepsilon_0} \\ \boldsymbol{\nabla} \cdot \mathbf{B} &= 0 \end{aligned} \tag{1.4}$$

where $\mathbf{J}$, ρ, μ_0, ε_0 and c represent the current density, the charge density, the magnetic permeability, the dielectric constant and the speed of light, respectively. These are the first-principle equations for collisionless plasma.

Ideally, the magnetosphere-ionosphere system should be modeled using the Vlasov equations together with Maxwell's equations. However, the computational complexity of direct Vlasov simulation is much higher than MHD. According to a recent research from Umeda *et al.* [9], Even with the fastest supercomputer in the world in 2014, it is impossible to use realistic physical parameters in their research. Reduction of the physical parameters is needed to enable a numerical simulation with a limited computer resource. High resolution run in 5D (or so-called 2.5D) Vlasov simulation of 1920×2560 grid points for two spatial dimensions, $60 \times 60 \times 60$ grid points for the three velocity dimensions requires 50 TB memory and 6,144 computational nodes (49,152 CPU cores) of the K computer. And it considering that a full six-dimensional global Vlasov simulation will require massive computing resources beyond Exascale which is left as a far future study. Therefore, MHD are commonly used as the basis of a numerical model. Although the MHD equations appear relatively simple at a first glance, the development of efficient and accurate numerical algorithms for their solution is a formidable task that has not yet come to conclusion.

In fluid descriptions of plasmas (MHD), the velocity distribution is not considered. This is achieved by replacing $f(\mathbf{r}, \mathbf{v}, t)$ with plasma moments such as number density n, flow velocity $\mathbf{u}$ and pressure $\mathbf{p}$ (Note that in order to avoid confusion, $\mathbf{u}$ is used for the flow velocity, where $\mathbf{v}$ is used for the velocity of the charged particles of the Vlasov equation). They are named plasma moments because the n-th moment of f can be

found by integrating $\mathbf{v}^n f$ over velocity.

$$\text{The zeroth moment (Density)} = n(\mathbf{r},t) = \int f(\mathbf{r},\mathbf{v},t)d\mathbf{v}$$
$$\text{The first moment (Momentum)} = n\mathbf{u}(\mathbf{r},t) = \int \mathbf{v}f(\mathbf{r},\mathbf{v},t)d\mathbf{v}$$
$$\text{The } n\text{-th moment } = \int \mathbf{v}^n f(\mathbf{r},\mathbf{v},t)d\mathbf{v}$$

These variables are only functions of position and time which can be treated as fluid. In two-fluid or multifluid model, the different species of particles are treated as different fluids with different densities, flow velocities and pressures. Taking the zeroth, first and second order moments of the Vlasov equation (Equation 1.3) and using the Maxwell's equations (Equation 1.4), full kinetics of pair-particles is neglected in collisionless plasma; instead, the hydrodynamic equations are obtained. The governing equations of MHD is the combination of the Navier-Stokes equations of fluid dynamics and Maxwell's equations of electromagnetics. The simplest form of MHD, the ideal MHD, assumes that the fluid has so little resistivity that it can be treated as a perfect conductor. This is the limit of infinite magnetic Reynolds number ($R_m = \mu\sigma LU$, where μ is the magnetic permeability, σ the electrical conductivity, L the typical length scale of the flow, U the typical velocity scale of the flow). The ideal MHD equations with the assumption of the magnetic permeability $\mu = 1$ can be expressed as hyperbolic system of conservation laws as follows [10]

$$\begin{aligned}
\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla}\cdot(\rho\mathbf{u}) &= 0 \\
\frac{\partial \rho\mathbf{u}}{\partial t} + \boldsymbol{\nabla}\cdot(\rho\mathbf{u}\mathbf{u} - \mathbf{B}\mathbf{B}) + \boldsymbol{\nabla}P^* &= 0 \\
\frac{\partial \mathbf{B}}{\partial t} - \boldsymbol{\nabla}\times(\mathbf{u}\times\mathbf{B}) &= 0 \\
\frac{\partial E}{\partial t} + \boldsymbol{\nabla}\cdot((E+P^*)\mathbf{u} - \mathbf{B}(\mathbf{B}\cdot\mathbf{u})) &= 0
\end{aligned} \tag{1.5}$$

The first equation is mass continuity. The second equation is the motion of an element of the fluid(The Navier-Stokes Equation). The third and forth equations are derived from the energy equation with Maxwell's equations. This is a closed set of eight nonlinear partial differential equations (PDEs) for the eight (2 scalar and 2 vector) variables. In ideal MHD equations, this is often accomplished with approximations to the heat flux through a condition of adiabaticity or isothermality. Here, ρ is the density, $\rho\mathbf{u}$ the momentum, $\mathbf{B}$ is the magnetic field, and E is the total energy. The total pressure is $P^* \equiv P + \frac{B^2}{2}$ where P is the gas pressure that satisfies the equation of state, $P \equiv$

$(\gamma-1)(E-\rho\frac{v^2}{2}-\frac{B^2}{2})$. The divergence-free constraint $\boldsymbol{\nabla}\cdot\mathbf{B}=0$ is satisfied by the flux constrained transport (CT) [11].

An important dimensionless number measures of a plasma effects to a magnetic field is called the plasma beta (β), and is defined as the ratio of the gas pressure (thermal energy density) P to the magnetic energy density $B^2/2\mu_0$.

$$\beta=\frac{2\mu_0 P}{B^2}. \tag{1.6}$$

When the magnetic field dominates in the fluid, $\beta \ll 1$, the fluid is forced to move along with the field. In the opposite case, when the field is weak, $\beta \gg 1$, the field is swirled along along by the fluid. In collisionless plasmas, It turns out that in both varieties of MHD the motion of the plasma parallel to magnetic field-lines is associated with the dynamics of sound waves, whereas the motion perpendicular to field-lines is associated with the dynamics of Alfvén waves. The MHD equations remain a reasonable approximation in a collisionless plasma in situations where the dynamics of sound waves, parallel to the magnetic field, are unimportant compared to the dynamics of Alfvén waves, perpendicular to the field. It is easily demonstrated that $\beta \sim (V_S/V_A)^2$, where V_S is the sound speed (thermal velocity), and V_A is the speed of an Alfvén wave. Thus, it obtains $\beta \ll 1$ which ensures that the collisionless parallel plasma dynamics are too slow to affect the perpendicular dynamics.

The resistivity term $\eta\nabla^2\mathbf{B}$ is neglected in the ideal MHD equations. implies that the magnetic field is tightly coupled to the fluid. In astrophysical systems R_m is usually very large because the scales are large (not because the resistivity is small). However, even if the resistivity is very low, MHD turbulence often produces very small-scale structures and large magnetic gradients, which lead to a finite rate of magnetic diffusion through the resistivity term, though the magnetic Reynolds number may be very big. When the fluid cannot be considered as completely conductive, but the other conditions for ideal MHD are satisfied, it is possible to use an extended model called resistive MHD (including the resistivity term). Resistive MHD describes magnetized fluids with finite electron diffusivity ($\eta \neq 0$). This diffusivity leads to a breaking in the magnetic topology; magnetic field lines can "reconnect" when they collide. Usually this term is small and reconnections can be handled by thinking of them are similar to shocks; this process has been shown to be important in the Solar-Earth magnetic interactions.

Shocks are transition layers across which there is a transport of particles, where the plasma properties change from one equilibrium state to another. The relation between the plasma properties on both sides of a shock (see Figure 1.3) can be obtained from the conservative form of the MHD equations. Information in MHD fluids is carried via

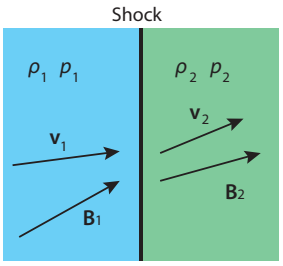

FIGURE 1.3: A schematic diagram of a shock wave situation with the density ρ, velocity $\mathbf{v}$, pressure p and magnetic field $\mathbf{B}$ indicated for each region.

three different waves — fast (compressional-Alfvén) waves, intermediate (shear-Alfvén) waves, and slow (magnetosonic) waves. Therefore, MHD fluids supports three different types of shock (fast-mode, intermediate and slow-mode shocks), corresponding to disturbances traveling faster than each of the those three types of waves. In general, a shock propagating through an MHD fluid produces a significant difference in plasma properties on either side of the shock front. The thickness of the shock is determined by a balance between convective and dissipative effects. However, dissipative effects in high temperature plasmas are only comparable to convective effects when the spatial gradients in plasma variables become extremely large. Hence, MHD shocks in such plasmas tend to be extremely narrow, and are well-approximated as discontinuous changes in plasma parameters. The MHD equations, and Maxwell's equations, can be integrated across a shock to give a set of jump conditions which relate plasma properties on each side of the shock front. If the shock is sufficiently narrow then these relations become independent of its detailed structure. The jump conditions across a time-independent MHD shock or discontinuity are referred as the Rankine-Hugoniot equations for MHD (These relations are often called Rankine-Hugoniot relations). In the frame moving with the shock or discontinuity, those jump conditions are as follows:

$$
\begin{aligned}
\rho_1 v_{n1} &= \rho_2 v_{n2} \\
B_{n1} &= B_{n2} \\
\rho_1 v_{n1}^2 + p_1 + \frac{B_{t1}^2}{2\mu_0} &= \rho_2 v_{n2}^2 + p_2 + \frac{B_{t2}^2}{2\mu_0} \\
\rho_1 v_{n1} \mathbf{v}_{t1} - \frac{\mathbf{B}_{t1} B_{n1}}{\mu_0} &= \rho_2 v_{n2} \mathbf{v}_{t2} - \frac{\mathbf{B}_{t2} B_{n2}}{\mu_0} \\
\left(\frac{\gamma}{\gamma - 1} \frac{p_1}{\rho_1} + \frac{v_1^2}{2} \right) \rho_1 v_{n1} &+ \frac{v_{n1} B_{t1}^2}{\mu_0} - \frac{B_{n1} (\mathbf{B}_{t1} \cdot \mathbf{v}_{t1})}{\mu_0} \\
= \left(\frac{\gamma}{\gamma - 1} \frac{p_2}{\rho_2} + \frac{v_2^2}{2} \right) \rho_2 v_{n2} &+ \frac{v_{n2} B_{t2}^2}{\mu_0} - \frac{B_{n2} (\mathbf{B}_{t2} \cdot \mathbf{v}_{t2})}{\mu_0} \\
(\mathbf{v} \times \mathbf{B})_{t1} &= (\mathbf{v} \times \mathbf{B})_{t2}
\end{aligned}
\tag{1.7}
$$

where ρ, $\mathbf{v}$, p, $\mathbf{B}$ are the plasma density, velocity, (thermal) pressure and magnetic field respectively. The subscripts t and n refer to the tangential and normal components of a vector with respect to the shock front. The subscripts 1 and 2 refer to the two states of the plasma on each side of the shock. Intermediate shocks are non-compressive — the plasma density does not change across the shock. Fast-mode and slow-mode shocks are compressive and are associated with an increase in entropy. The tangential component of the magnetic field increases across the fast-mode shock. In the opposite case, it decreases across slow-mode shock. The type of shocks depend on the relative magnitude of the upstream velocity in the frame moving with the shock with respect to some characteristic speed. Those characteristic speeds, the slow and fast magnetosonic speeds, are related to the Alfvén speed, V_A and the sound speed, V_S as follows:

$$\begin{aligned} a^2_{\text{fast}} &= \frac{1}{2}\left[\left({V_S}^2 + {V_A}^2\right) + \sqrt{\left({V_S}^2 + {V_A}^2\right)^2 - 4{V_S}^2{V_A}^2\cos^2\theta_{Bn}}\right] \\ a^2_{\text{slow}} &= \frac{1}{2}\left[\left({V_S}^2 + {V_A}^2\right) - \sqrt{\left({V_S}^2 + {V_A}^2\right)^2 - 4{V_S}^2{V_A}^2\cos^2\theta_{Bn}}\right] \end{aligned} \tag{1.8}$$

where θ_{Bn} is the angle between the incoming magnetic field and the shock normal vector. The normal component of the slow shock, intermediate shock and fast shock propagate with velocity a_{slow}, V_{An} and a_{fast}, respectively, and in the frame moving with the upstream plasma. The fast mode waves have higher phase velocities than the slow mode waves because the density and magnetic field are in phase, whereas the slow mode wave components are out of phase. The Earth's bow shock, which is the boundary where the solar wind's speed drops due to the presence of the Earth's magnetosphere, is a fast mode shock.

Although the MHD equations are often under scrutiny when applied to space plasmas, experience and many successful approaches have proven that they are adequate in many situations where the spatial scale of interest is larger than the ion gyroradius ($r_i = v_{Ti}/\omega_{ci}$, where v_{Ti} is the ion trapping rate and ω_{pi} the ion gyrofrequency) and the ion inertial scales (the scale at which ions decouple from electrons and the magnetic field becomes frozen into the electron fluid rather than the bulk plasma, $d_i = c/\omega_{pi}$, where c is the speed of light and ω_{pi} the ion plasma frequency), and the temporal scale is longer than the ion gyroperiod. In assessing the validity of the MHD equations one must consider that they are conservation equations. Specifically, MHD describes the conservation of mass, momentum, energy, and magnetic flux. As far as the plasma is concerned the only significant underlying assumption is that the velocity distribution functions of the plasma constituents are only a function of $|v - v_d|$ in phase space, where v_d is the drift speed (first moment of the distribution). This is trivially fulfilled for

a Maxwellian distribution and causes all moments higher than the scalar pressure to vanish, or at least to decouple [12]. Violations of the $f(v - v_d) = f(|v - v_d|)$ assumption are mostly mild in large parts of the magnetosphere.

MHD equations can be used in modelling phenomenon in a wide range of applications including astrophysics [13, 14], and space plasmas [15, 16]. For example, 3D MHD simulations have been widely adopted in space weather simulations. The historical review and current status of the existing popular 3D MHD models can be found in [17, 18].

Global MHD model has been used in modeling various phenomenon in space physics since it was developed by Leboeuf *et al.* [19]. The first global MHD model was a two-dimensional (2D) model and the same group then developed the first three-dimensional (3D) model [20]. The physical and numerical foundation of global MHD models can be found in [21]. Global MHD simulation is useful in investigating the the features of the magnetosphere [22]. Ogino *et al* [7, 23, 24] developed the global MHD simulation model to simulate and investigate the solar wind interaction with the Earth's magnetosphere. This has been widely used in investigating the solar wind-planetary magnetosphere interaction, such as the Jupiter's magnetosphere [25–28] and Saturn's magnetosphere [29–32]. Other applications of MHD simulation of the solar wind are interaction with Mercury [33] which has dipole as the Earth but very weak. In recent year, the simulation of the lunar wake [34] which does't have a dipole field was also presented.

The wide use of MHD simulation shows it is applicability to planet's magnetosphere. However, single fluid MHD is not sufficient and is constantly being improved, modified and applied to particular studies. For example, MHD has difficulties in magnetic reconnection since it ignores the electrons. Therefore, Hall MHD is generally used in simulations of reconnection [35, 36]. Extensions of multispecies MHD simulation was used in investigating the crustal fields of Mars [37]. In the past, due to the lack of computational power, scientists prefer to use the most simple MHD with resistivity for space plasma problems. Nowadays, the computational power is much improved. More accurate schemes, modification or extensions of MHD is becoming more popular. Feng *et al* have adapted the conservation element/solution element method (CESE) method to MHD with curvilinear coordinates and applied it to the three-dimensional MHD simulations of the interaction between the solar wind and Saturn's magnetosphere. However, to the extent of the author's knowledge, almost all enhancements of MHD simulations focus on the details of the internal structure of the magnetosphere. For the planet's magnetosphere, only the region surrounding the planet is being simulated. As a result, there are many ad hoc settings that had to be added to perform a correct simulation. For example, according to the simulation model proposed by Ogino *et al.* the shear

boundary condition is used at the y and z boundaries which are in the region within the bow shock and the magnetosheath. Therefore, only a few number of simulations cover the wide region and investigate the variation of the whole structure of the magnetosphere as well as the bow shock [38, 39]. The bow shock covers a wide region. So enlarging the domain to cover the whole bow shock requires a large number of grid points, which leads to high memory usage and computational time. Scientists prefer to allocate their limited resource to increase the resolution or the accuracy of the simulation. In fact, the simulation domains of [38, 39] are very limited. Beside, only a 1/4 of the magnetosphere is simulated in [40]. This gives the motivation of not only enhancing the performance, but also to save the memory usage so that one may enlarge the simulation domain to investigate the whole mangetosphere.

1.4 GPU Computing

Computing power is a critical resource in numerical simulation, especially at large-scale. Large-scale high resolution simulations require huge computational power and usually needs to be run on supercomputers (cluster) for hours, days or even weeks to get the result. 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 why 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.

1.4.1 GPUs and GPGPU

GPUs were originally a graphics processing hardware, providing high parallelism to accelerate the calculations in computer graphics. GPU has higher floating-point capability and bandwidth compared to CPU, and they are being increased faster than CPU as shown in Figures 1.4 and 1.5. The reason is that the GPU is specialized for compute-intensive, highly parallel computation and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control, as schematically illustrated in Figure 1.6. To exploit of the high parallelism and high GFLOPS (Giga Floating-point Operations Per Second), GPUs had been used for general purpose computations since the very first programmable GPUs appeared in 2001.

The programmable shading language — shader, was introduced to allow GPUs to run user-defined shading processes(program). However, a shader runs as an alternative to the default shading process in the graphics pipeline. Therefore, it is a difficult task to use GPUs for computational purposes because we still have to use the shader for the computations and the program flow needs to follow the rule(default settings) of the graphics pipeline, such as the coordinate system, and only floating point number(single precision and half precision) is supported. In addition, data passed as the input to be processed in a shader has to be stored in texture memory. We have to retrieve the results from the framebuffer or from texture with a render–to–texture technique. In 2006, Compute Unified Device Architecture (CUDA) [2] was introduced, where GPUs could be used for computation without following the graphics pipeline. Following that, compute purpose Tesla GPUs impacted the fields related to high performance computing (HPC) and had became a common device in the nodes of a cluster. Nowadays, not only the graphics related fields such as image processing and computer vision take advantages of the parallelism of GPUs, but also Artificial Intelligence (AI) and numerical simulations. In fact, scientific computing related to astrophysics and space simulation were the very first users that benefited from the acceleration of GPGPU and GPU computing, due to their high computational intensity. GPGPU and GPU computing also influence the development of GPUs. New features such as GPU-Direct of the Fermi generation GPUs for faster communication between multiple GPUs and the Hyper-Q to increase the parallelism of multi-streaming of the Kepler generation GPUs had been proposed for better efficiency in scientific computation using GPUs.

1.4.2 Distributed Multi-GPU Systems

GPUs have become commonplace in the high performance computing related areas including numerical simulations. In recent years, an increasing number of work about numerical simulations on GPUs have been presented [41–46]. GPUs not only provide very high FLOPS (floating-point operations per second), but the energy efficiency measured by FLOPS per Watt, is also impressive.The capacity of video memory (GRAM, device memory) on a single GPU is limited. Thus, multi-GPU systems are needed for large scale computation. Nowadays, many high ranking supercomputers in the TOP500 [47] of GREEN500 [48] lists contain GPUs as acceleration devices to speedup the computation.

The GPU-rich supercomputer TSUBAME (Tokyo Tech Supercomputer and Ubiquitously Accessible Mass Storage Environment, Figure 1.7) is one of the most representative GPU supercomputers in the world. It began operating as TSUBAME 1.0 in 2006, after upgrading to TSUBAME 1.2, in 2010, TSUBAME was upgraded to TSUBAME 2.0

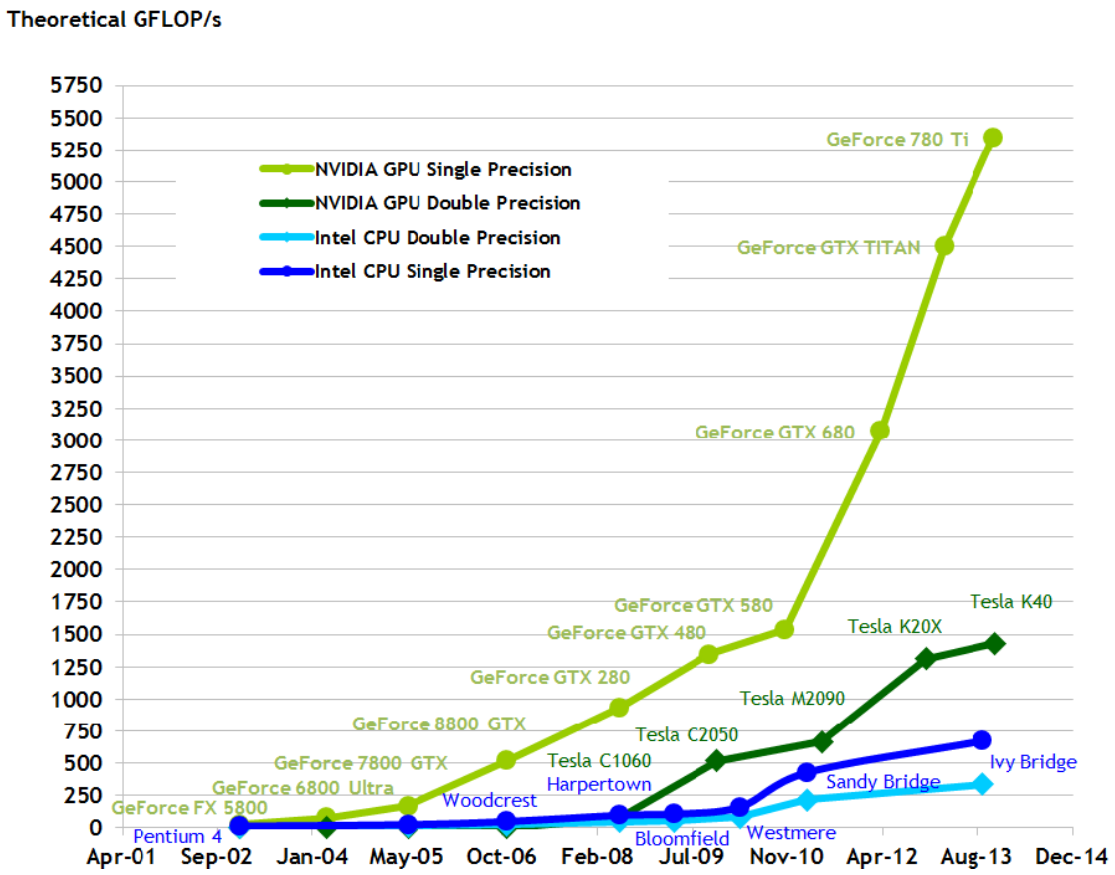

FIGURE 1.4: Recent development of GPUs and CPUs (FLOPS)[2].

in November 2010. TSUBAME 2.0 consists of 1,408 Hewlett-Packard Proliant SL390s nodes and each node is equipped with three NVIDIA Tesla M2050 (Fermi architecture) GPUs (see Figure 1.8), thus the total number of GPUs is 4,224. TSUBAME 2.0 was the first petaflop(PFLOP) supercomputer in Japan, at 2.4 PFLOPS theoretical performance. TSUBAME 2.0 also achieved world-leading power efficiency as No. 2 in the Green500 rankings for supercomputer energy efficiency. Several interesting applications have been implemented on TSUBAME 2.0, for examples, ASUCA model for weather simulation [42], phase-field simulation for dendritic solidification [43, 49], and lattice-Boltzmann based incompressible flow computation [44]. In the fall of 2013, all the Fermi GPUs of the computing nodes of TSUBAME had been upgraded to the next generation Kepler GPUs. Theoretical peak performance has increased to 17.1 PetaFLOPS in single precision, which is 3.6× compared to TSUBAME 2.0. On the other hand, TSUBAME 2.5 uses 20 percent less electrical power than TSUBAME 2.0.

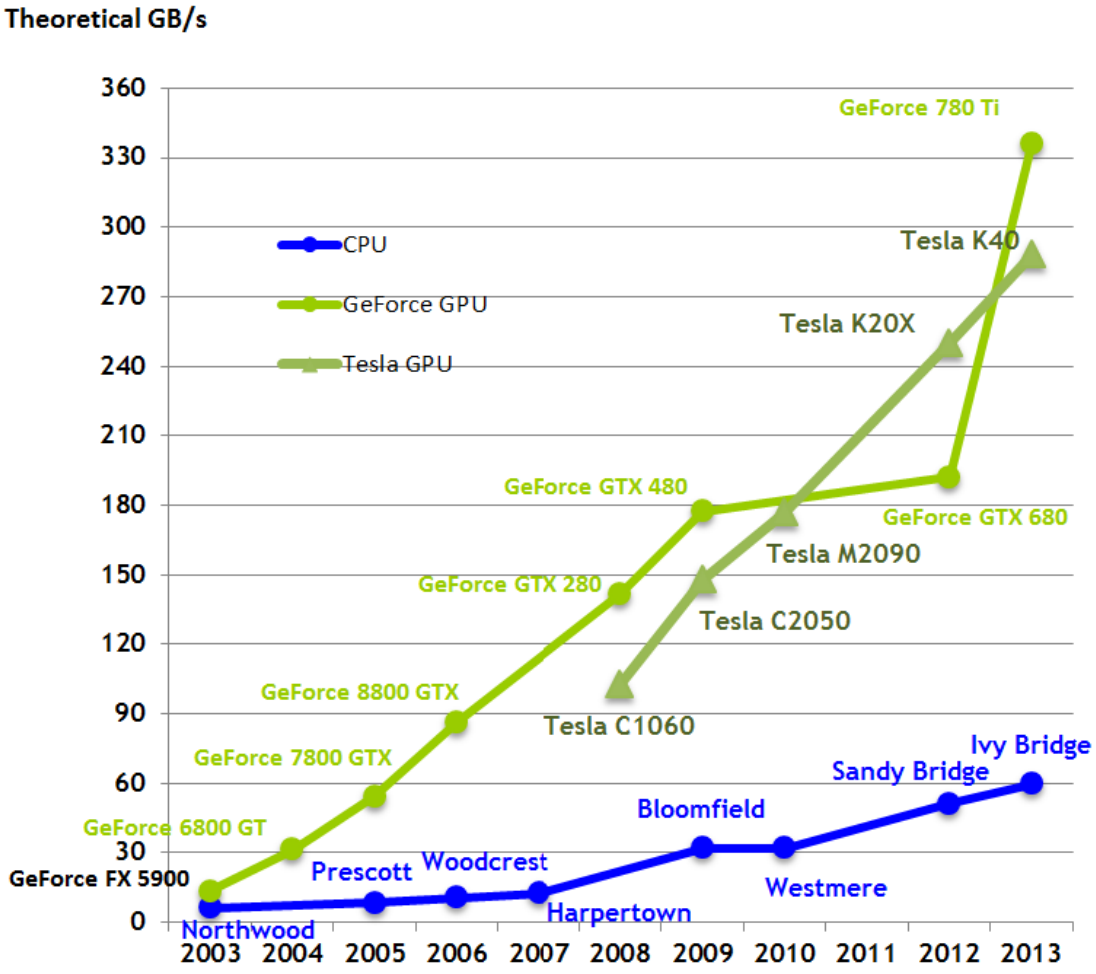

FIGURE 1.5: Recent development of GPUs and CPUs (bandwidth)[2].

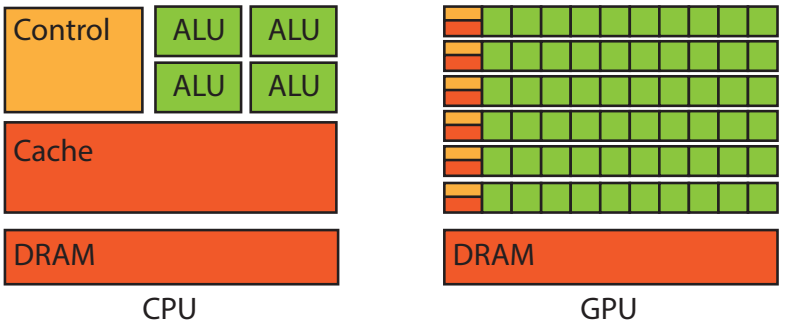

FIGURE 1.6: The GPU devotes more transistors to data processing[2].

FIGURE 1.7: The TSUBAME supercomputer [3].

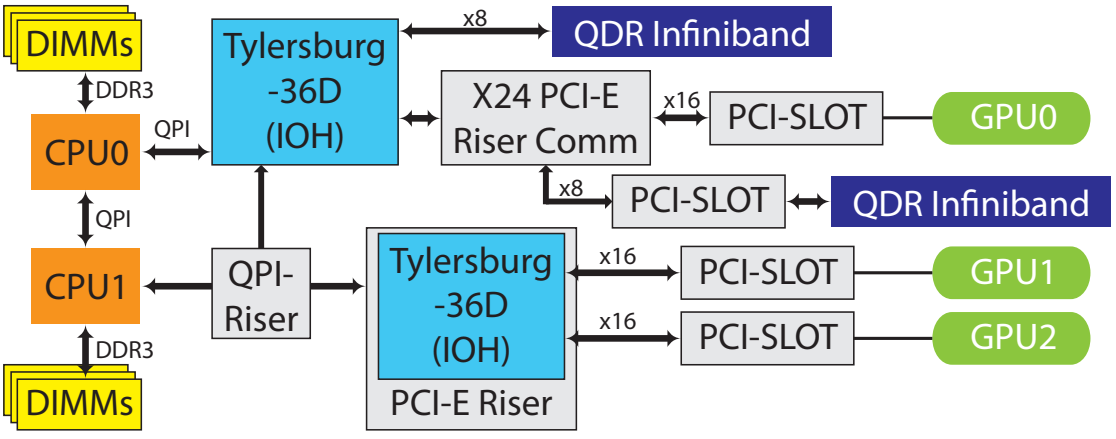

FIGURE 1.8: The configuration of a calculation node of the TSUBAME supercomputer [3].

Chapter 2

GPU Accelerated Space Plasma Simulations

2.1 Plasma Simulations on GPUs

Modern graphics processing units (GPUs) have been gaining popularity as a accelerator device in high performance scientific computing for that past decade, since the advent of NVIDIA Compute Unified Device Architecture (CUDA) [2, 50]. More and more numerical simulations in different fields have been performed on GPUs. Numerical simulations of astrophysics and space science, whose computational complexity are generally very high, were the very first research field to utilize GPUs to accelerate the simulation process.

In recent years, an increasing number of work about plasma simulations (particle-in-cell (PIC) and MHD models) on GPUs have been presented, though most of them work on a single GPU. The early work of Stantchev et al. [51, 52] implemented a PIC code on GPUs for plasmas simulations and visualizations. The code demonstrated a speedup of 11-22 for different grid sizes on a NVIDIA GeForce 8800 Ultra graphics card. Abreu et al. [53] developed a 2D relativistic GPU PIC code and this code can perform simulations on a NVIDIA Tesla C1060 graphics card and direct visualizations on a NVIDIA Quadro FX 1800 graphics card. Kong et al. [54] developed a 2D3V fully relativistic electromagnetic code on a NVIDIA GeForce GTX 280 graphics card and achieved speedup of 81x and 27x over an Intel Core 2 Duo E7200 2.53 GHz CPU using only a single core for cold plasma runs and extremely relativistic plasma runs, respectively. Decyk and Singh [55] developed a new parameterized PIC algorithm and data structure on a NVIDIA GeForce GTX 280 graphics card. They reported speedups of about 15-25 compared to an Intel Nehalem 2.66 GHz processor for a simple 2D electrostatic code. Madduri et

al. [56] reported the first study on tuning gyrokinetic PIC algorithms for GPUs, using a NVIDIA C2050 graphics card. For multi-GPU implementations, Burau et al. [41] presented one of the first multi-GPU PIC implementation called *PIConGPU*, which is a fully relativistic, 2D electromagnetic code for simulating the acceleration of electrons in an under-dense plasma by a laser-driven wakefield. Bastrakov et al. [45] presented an approach to high-performance implementation of PIC algorithms on heterogeneous cluster systems.

There are also some implementations of MHD simulations on GPUs, though most of them only utilize a single GPU. Our previous work *GPU-MHD* code [57] implemented a total variation diminishing (TVD) algorithm on a NVIDIA GeForce GTX 290 graphics card and achieved a speedup of 84 times in 3D over an Intel Core i7 965 3.2 GHz CPU. Pang et al. [58] also implemented the same algorithm on NVIDIA GTX 260 graphics card and reached a speedup of 105 times in 3D over a Xeon E5506 2.13 GHz CPU. Wang et al. [59] implemented a compressible inviscid fluid solver and extended it to support MHD simulations on a NVIDIA Quadro FX 5600 graphics card. The code achieved a factor of ten speedup over a 3 GHz CPU. They also tested the code on a small cluster with four nodes and each has a NVIDIA GeForce 8800 GT graphics card. They reported a close ideal speedup for up to four GPUs. Ueda et al. [60] implemented the CIP-MOCCT method on a NVIDIA GeForce GTX 480 graphics card and obtained speedups of 30 over a Intel i7 2.93 GHz CPU. Zink [61] developed a GPU-accelerated general relativistic MHD code *HORIZON* and tested it on a NVIDIA GeForce GTX 580 graphics card and two NVIDIA Tesla C2070 graphics cards installed in a workstation. Lin et al. [62] presented the large-scale high-Lundquist number reduced MHD simulations of the solar corona on GPU accelerated distributed memory machines, but a very limited amount of implementation details were reported in Lin's work. Recently, Wasiljew and Murawski [63] presented a GPU implementation of the 2D Athena code [14] and tested it on a NVIDIA GeForce GTX 460 graphics card. Although there are quite a large number of GPU MHD implementations available, no optimization techniques dedicated to improve the performance of MHD simulations on distributed multi-GPU systems have been presented in detail.

2.2 The GPU-MHD Code

In this section, we briefly review our *GPU-MHD* code for ideal MHD simulations on a single GPU. This work had been published in [57]. *GPU-MHD* was designed to be run entirely on the GPU and it supports both single and double precision. Here we summarize our strategies for the design as follows:

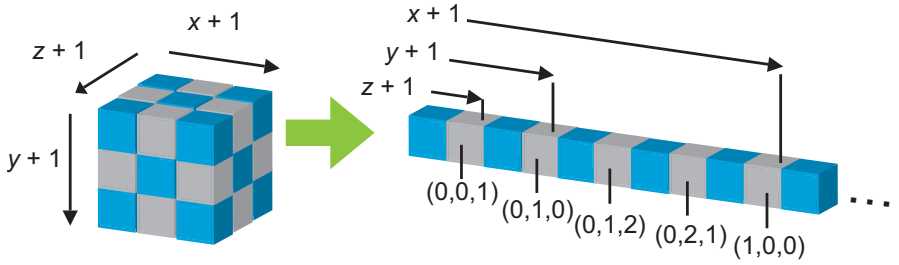

FIGURE 2.1: Mapping between 1D array and 3D array.

- One or more CUDA kernels were designed and implemented for each step of the numerical scheme to exploit the parallelism of GPUs;
- Storage of the intermediate results (such as the "flux" and some interpolated values of each grid point), which will not be used in the next calculation step, are reused to reduce memory usage;
- In order to provide effective memory access, the eight components (ρ, ρv_x, ρv_y, ρv_z, E, B_x, B_y, B_z) of the MHD equations are stored in separate arrays and each component of a grid point is stored close to the same component of the neighboring grid points. In addition, only the necessary component of a calculation kernel will be accessed.

For memory arrangement of *GPU-MHD*, we store the data in one-dimensonal arrays and perform the parallel computation with one-dimensional threads due to the limited support of multidimensonal threads by the early version of CUDA.

Figure 2.1 shows the data storage arrangement of *GPU-MHD*. The storage and threading method can be extended to solve multidimensional problems using an simple index calculation shown in Equations 2.1 and 2.2.

$$\begin{cases} INDEX_x &= index \quad / \quad (SIZE_y \times SIZE_z) \\ INDEX_y &= [index \ \textbf{mod} \ (SIZE_y \times SIZE_z)]/SIZE_z \\ INDEX_z &= index \ \textbf{mod} \ SIZE_z \end{cases} \tag{2.1}$$

$$\begin{aligned} INDEX_x \pm 1 &= index \pm (SIZE_y \times SIZE_z) \\ INDEX_y \pm 1 &= index \pm SIZE_z \\ INDEX_z \pm 1 &= index \pm 1 \end{aligned} \tag{2.2}$$

Here $INDEX_x$, $INDEX_y$, and $INDEX_z$ are the indexes of a 3D matrix. $index$ is the 1D index used in the 1D array (storage) of *GPU-MHD*, $SIZE_y$, and $SIZE_z$ are the

matrix size (number of grid points in our case) of a 3D matrix. Equation 2.1 expresses the mapping of three-dimensional indexes to one-dimensional indexes. Equations 2.2 is the shift operations. Shift operations are very important in numerical solution because it is necessary to find out the neighboring grid points in the stencil calculations.

A second-order accurate (in space and time) total variation diminishing (TVD) method [64] is used for solving the ideal MHD equations (Equation 1.5). In this method, we first hold the magnetic field fixed and then update the fluid variables. Then we perform a reverse procedure to complete one time step. The three dimensional problem is solved by splitting it into one-dimensional sub-problems using a Strang-type directional splitting [65]. Here we summarize the numerical method. The ideal MHD equations can be written in flux-conservative vector form by considering the advection along the x direction as follows

$$\frac{\partial \boldsymbol{u}}{\partial t} + \frac{\partial \boldsymbol{F}(u)}{\partial x} = 0 \tag{2.3}$$

where the conserved fluid variables and the flux vectors are given by

$$\boldsymbol{u} = \begin{pmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \rho v_z \\ E \end{pmatrix}, \quad \boldsymbol{F} = \begin{pmatrix} \rho v_x \\ \rho v_x^2 + P^* - B_x^2 \\ \rho v_x v_y - B_x B_y \\ \rho v_x v_z - B_x B_z \\ (E + P^*) v_x - B_x \mathbf{B} \cdot \mathbf{v} \end{pmatrix} \tag{2.4}$$

Equation A.1 is solved by Jin and Xin's relaxing TVD method [66]. The fluxes are obtained using a monotone upwind scheme for conservation laws (MUSCL) with a second-order TVD (van Leer limiter) correction. A second-order Runge-Kutta scheme is used for the time integration. Let L_i be the update operator of $\boldsymbol{u}^t$ to $\boldsymbol{u}^{t+\Delta t}$ including the flux along the i-direction with time step Δt. Each L_i includes three update operations in sequence, for example, L_x includes the fluid update along x, the update of B_y along x, and the update of B_z along x. A forward sweep and a reverse sweep are defined as $\boldsymbol{u}^{t+\Delta t} = L_z L_y L_x \boldsymbol{u}^t$ and $\boldsymbol{u}^{t+2\Delta t} = L_x L_y L_z \boldsymbol{u}^{t+\Delta t}$, respectively. Then the dimensional splitting of the relaxing TVD can be expressed as the follows:

$$\begin{aligned} \boldsymbol{u}^{t_2} &= \boldsymbol{u}^{t_1+2\Delta t_1} = L_x L_y L_z L_z L_y L_x \boldsymbol{u}^{t_1} \\ \boldsymbol{u}^{t_3} &= \boldsymbol{u}^{t_2+2\Delta t_2} = L_z L_x L_y L_y L_x L_z \boldsymbol{u}^{t_2} \\ \boldsymbol{u}^{t_4} &= \boldsymbol{u}^{t_3+2\Delta t_3} = L_y L_z L_x L_x L_z L_y \boldsymbol{u}^{t_3} \end{aligned} \tag{2.5}$$

where Δt_1, Δt_2, and Δt_3 are sequential time steps after each double sweep. The operator L_i contains the majority of the calculations in the numerical method. Each sweeping operation L_i will update both the fluid variables and orthogonal magnetic fields along the i-direction. Since the updates in L_i are first performed along the x-direction, then y-direction, and then z-direction, the numerical kernel only has to be implemented once and can be used for all directions by changing the targeting index. The calculation steps in *GPU-MHD* are shown as the follows, The corresponding flow chart is shown as Figure 2.2:

1. CUDA/OpenGL initialization
2. Setup the initialize condition for the specified MHD problem:
   $\boldsymbol{u} = (u_1, u_2, u_3, u_4, u_5)$ of all grid points, $\mathbf{B} = (B_x, B_y, B_z)$ of cell faces, and set parameters such as time t, etc.
3. Copy the the initialize condition $\boldsymbol{u}$, $\mathbf{B}$ to device memory (CUDA global memory)
4. For all grid points, calculate the c_{max} by Equation A.17 (implemented with a CUDA kernel)
5. Use `cublasIsamax` (in single precision mode) function or `cublasIdamax` (in double precision mode) function of the CUBLAS library to find out the maximum value of all c_{max}, and then determine the Δt
6. Since the value of Δt is stored in device memory (GRAM), read it back to host memory (RAM)
7. Sweeping operations of the relaxing TVD (Calculation of the $L_i, i = x, y, z$)
8. $t = t + 2\Delta t$
9. `If` t reaches the target time, go to next step
   `else` repeats the procedure from step (4)
10. Read back data $\boldsymbol{u}$, $\mathbf{B}$ to host memory
11. Output the result

Implementing parallel computation using CUDA kernels is somewhat similar to parallel implementation on a CPU-cluster, but it is not the same. The major concern is the memory constrain in GPUs. CUDA makes parallel computation process on GPUs which can only access their graphics memory (GRAM). Therefore, data must be stored in GRAM in order to be accessed by GPUs. There are several kinds of memory on graphics hardware including registers, local memory, shared memory, and global memory, etc.,

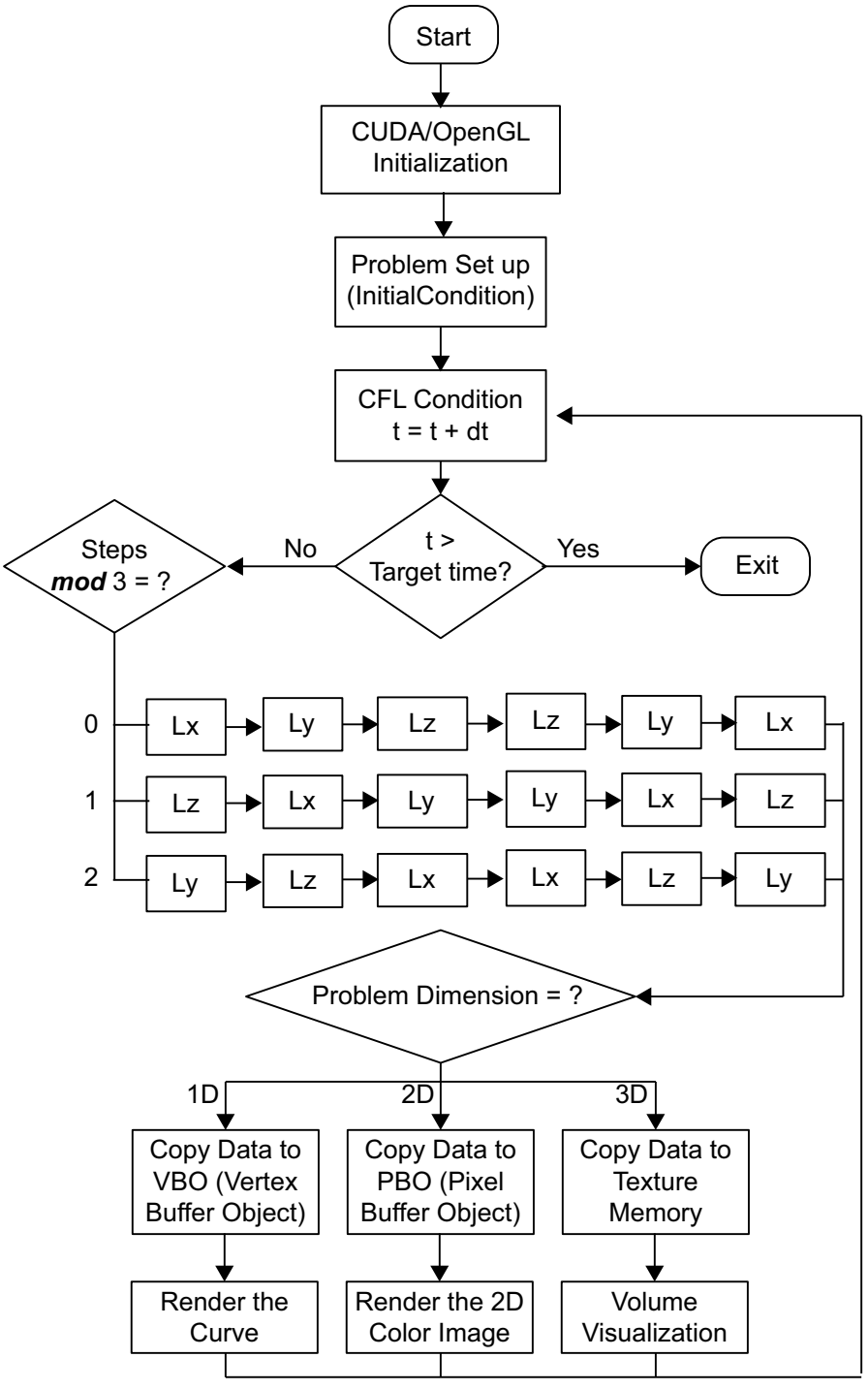

FIGURE 2.2: The flowchart of *GPU-MHD* .

and they have different characteristics and usages [2], making memory management of CUDA quite different compared to parallel computation on a CPU-cluster. In addition, even the size of GRAM in a graphics card increases rapidly in newer models, but not all the capacity of GRAM can be used to store data arbitrarily. Shared memory and local memory are flexible to be used, however, their sizes are very limited in a block and thus they cannot be used for storing data with large size. In general, numerical solution of conservation laws will generate many intermediate results (for example, $\boldsymbol{u}^{t+\Delta t/2}$, $\boldsymbol{F}$, c, w, etc.) during the computation process, these results should be stored for subsequent steps in the process. Therefore, global memory was mainly used in *GPU-MHD*.

After the c_{max} in Equation A.17 is found, we can get the Δt by determining the Courant-Number (cfl). The sequential step is the calculation of L_i ($i = x, y, z$). The implementation of L_i includes two parts: update the fluid variables and update the orthogonal magnetic fields. The first part of the L_i calculation process is named $fluid_i$ for the following description. The fluid variables will be updated along x-direction. **Algorithm 1** shows the steps and GPU kernels of the processes of $fluid_x$ (the data of $\boldsymbol{u}$ and $\mathbf{B}$ are already copied to device memory), all the steps are processed on all grid points with CUDA kernels in parallel.

Algorithm 1 Algorithm of $fluid_x$, all equations and difference calculations are processed using CUDA kernels

1: Load $\boldsymbol{u}$, $\mathbf{B}$ and Δt
2: Memory allocation for the storage of the intermediate results: $\mathbf{B_{temp}}$, $\boldsymbol{u}_{temp}$, $\mathbf{flux_{temp}}$, $\mathbf{other_{temp}}$, ($\mathbf{other_{temp}}$ includes the storage of $\boldsymbol{F}$, c, w, etc)
3: $\mathbf{B_{temp}} \leftarrow$ results obtained by Equation A.18 using $\mathbf{B}$
4: $\mathbf{other_{temp}} \leftarrow$ results obtained by Equations A.8 and A.10 using $\boldsymbol{u}$
5: $\mathbf{flux_{temp}} \leftarrow$ the flux of a half time step: difference calculation obtained by Equation A.15 using $\mathbf{other_{temp}}$
6: $\boldsymbol{u}_{temp} \leftarrow$ calculate the intermediate result ($\boldsymbol{u}^{t+\Delta t/2}$) using Equation A.15 using $\boldsymbol{u}$ and $\mathbf{flux_{temp}}$
7: $\mathbf{other_{temp}} \leftarrow$ results obtained by Equations A.8 and A.10 using $\boldsymbol{u}_{temp}$
8: $\mathbf{flux_{temp}} \leftarrow$ the flux of another half time step: difference calculation obtained by Equation A.16) and the limiter (Equation A.13) using $\mathbf{other_{temp}}$
9: Calculate the result of $\boldsymbol{u}^{t+\Delta t}$ using $\mathbf{flux_{temp}}$ using Equation A.16 and save it back to $\boldsymbol{u}$
10: Free the storage of the intermediate results
11: (Continue to the second part of L_x, update the orthogonal magnetic fields)

In this process, we have to calculate the magnetic fields of the grid point (Equation A.18) first because all the magnetic fields are defined on the faces of the grid cell [64]. To update the fluid variables of L_i, the main process, which includes one or even several CUDA kernels, is to calculate the affect of the orthogonal magnetic fields to the fluid variables. After the above processes of flux calculation is processed, the value of fluid — $\boldsymbol{u}$ will be updated from $\boldsymbol{u}^t$ to $\boldsymbol{u}^{t+\Delta t}$.

The second part of the L_i calculation process is to update the orthogonal magnetic fields (For example, magnetic fields in y-dimension ($B_{y\to x}$), and z-dimension ($B_{z\to x}$) will be updated after the $fluid_x$ of L_x is finished). **Algorithm 2** shows the calculation steps of the magnetic fields.

Algorithm 2 Algorithm of ($B_{y\to x}$) and ($B_{z\to x}$), all equations and difference calculations are processed using CUDA kernels

1: (After the processes of fluid, we obtain an updated $\boldsymbol{u}$)
2: Load u_1 (density ρ), u_2 (ρv_x), $\mathbf{B}$ and Δt
3: Memory allocation for the intermediate results: $\mathbf{B_{temp}}$, $\mathbf{flux_{temp}}$, $\mathbf{vx_{temp}}$ and $\mathbf{vx_{face}}$
4: $\mathbf{vx_{temp}} \leftarrow$ determine the fluid speed with the updated u_1 and u_2 in $fluid_x$, with the difference calculated in y-dimension
5: $\mathbf{vx_{face}} \leftarrow$ Results obtained by Equation A.19
6: $\mathbf{flux_{temp}} \leftarrow$ the flux of a half time step: difference calculation of "flux of magnetic field in y-dimension" obtained by Equations A.15 and A.20)
7: $\mathbf{B_{temp}} \leftarrow$ calculate the intermediate result ($\boldsymbol{u}^{t+\Delta t/2}$) by applying Equation A.15 to B_y (not by applying Equation (A.15) to $\boldsymbol{u}$) with B_y and $\mathbf{flux_{temp}}$
8: $\mathbf{flux_{temp}} \leftarrow$ the flux of another half time step: difference calculation obtained by Equation A.15, the limiter of Equation A.13 and Equation (A.20)
9: Calculate the result of $B_x^{t+\Delta t}$ and $B_z^{t+\Delta t}$ with $\mathbf{flux_{temp}}$ by applying Equation A.16) to B_y, and save it back to $\mathbf{B}$
10: (the following steps is similar to above steps but the affected orthogonal magnetic field is changed from y to z)
11: $\mathbf{vx_{temp}} \leftarrow$ determine the fluid speed with the updated u_1 and u_2 in $fluid_x$, with the difference calculated in z-dimension
12: $\mathbf{vx_{face}} \leftarrow$ Results obtained with Equation A.19 using index of i, j, $k+1/2$
13: $\mathbf{flux_{temp}} \leftarrow$ the flux of a half time step: difference calculation of "flux of magnetic field in z-dimension" obtained by Equations A.15 and (A.20)
14: $\mathbf{b_{temp}} \leftarrow$ calculate the intermediate result ($\mathbf{u}^{t+\Delta t/2}$) by applying Equation A.15 to B_z (not by applying Equation A.15 to $\boldsymbol{u}$) with B_z and $\mathbf{flux_{temp}}$
15: $\mathbf{flux_{temp}} \leftarrow$ the flux of another half time step: difference calculation obtained by Equation A.15, the limiter of Equation A.13 and Equation A.20
16: Calculate the results of $B_x^{t+\Delta t}$ and $B_z^{t+\Delta t}$ with $\mathbf{flux_{temp}}$ by applying Equation A.16) to B_z, and save it back to $\mathbf{B}$
17: Free the storage of the intermediate results

After one whole process of L_x is processed, both fluid and magnetic fields are updated to $t+\Delta t$ with the affect of the flow in x-dimension. One whole sweeping operation sequence includes two L_x, L_y, and L_z (see Equations A.22). So we actually get the updated fluid and magnetic fields of $t+2\Delta t$ after one sweeping operation sequence. As the only difference among L_x, L_y, and L_z is the dimensional index, we only need to change the targeting index to get a generic sweeping operator L_i to work for a particular dimension by performing indexing operations in all L_i kernels. Figure 2.3 shows the flux computation of *GPU-MHD*. Once the one step of the whole computation pipeline in Figure 2.2 is completed, the MHD simulation results will be stored in graphics memory

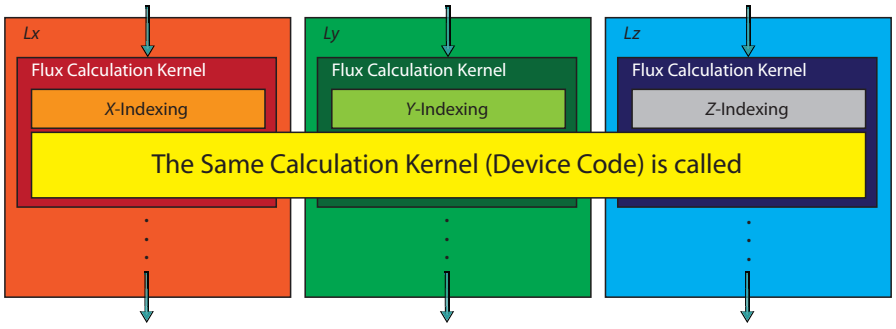

FIGURE 2.3: Flux computation in *GPU-MHD*. The same kernel is called in L_i with different indexing for the calculation in different direction.

(GRAM). These results are ready for real-time visualization or copy back to the CPU for recording the data of other usages.

GPU-MHD provides different visualization methods for one-dimensional, two-dimensional and three-dimensional problems. Since the simulation data are stored in the GRAM of the GPU board. It can be visualized directly using the graphics features of the GPU. Referring to the flowchart of Figure 2.2, the simulation results of each step will be visualized with different operations, depending on the dimension of the simulation. For GPUs doesn't support graphics features, or if the user prefer not to visualize the result in real-time (disable the real-time visualizatoin), visualization steps after L_i will be skipped. To visualize one-dimensional problems for each time step, the simulation results are copied to the CUDA global memory that mapped to the Vertex Buffer Object (VBO) [67]. For all grid points, one grid point is mapped to one vertex. The position of each grid point is mapped as the x-position of the vertex and the selected physical value (ρ, p, etc.) is mapped as the y-position of the vertex. Then a curve of these vertices is drawn. Since the VBO is mapped to CUDA global memory and simulation results are stored in GRAM, the copying and mapping operations are fast. Experimental result shows that *GPU-MHD* with real-time visualization can achieve 60 frame per second (FPS) in single precision mode and 30 FPS in double precision mode.

The operational flow of visualization of 2D problems is similar to that in 1D visualization. However, instead of Vertex Buffer Object (VBO), Pixel Buffer Object(PBO) [67] is used. For each time step, the simulation results are copied to the CUDA global memory that are then mapped to the PBO. For all grid points, one grid point is mapped to one pixel. The x and y position of each grid point are mapped as the corresponding x-position and the y-position of the vertex and the selected physical value (ρ, p, etc.) is mapped as the color of the pixel to form a color image. To render this color image, a transfer function is set to map the physical value to the color of the pixel and then the resulting image is drawn. Similar to VBO, PBO is also mapped to CUDA global memory and

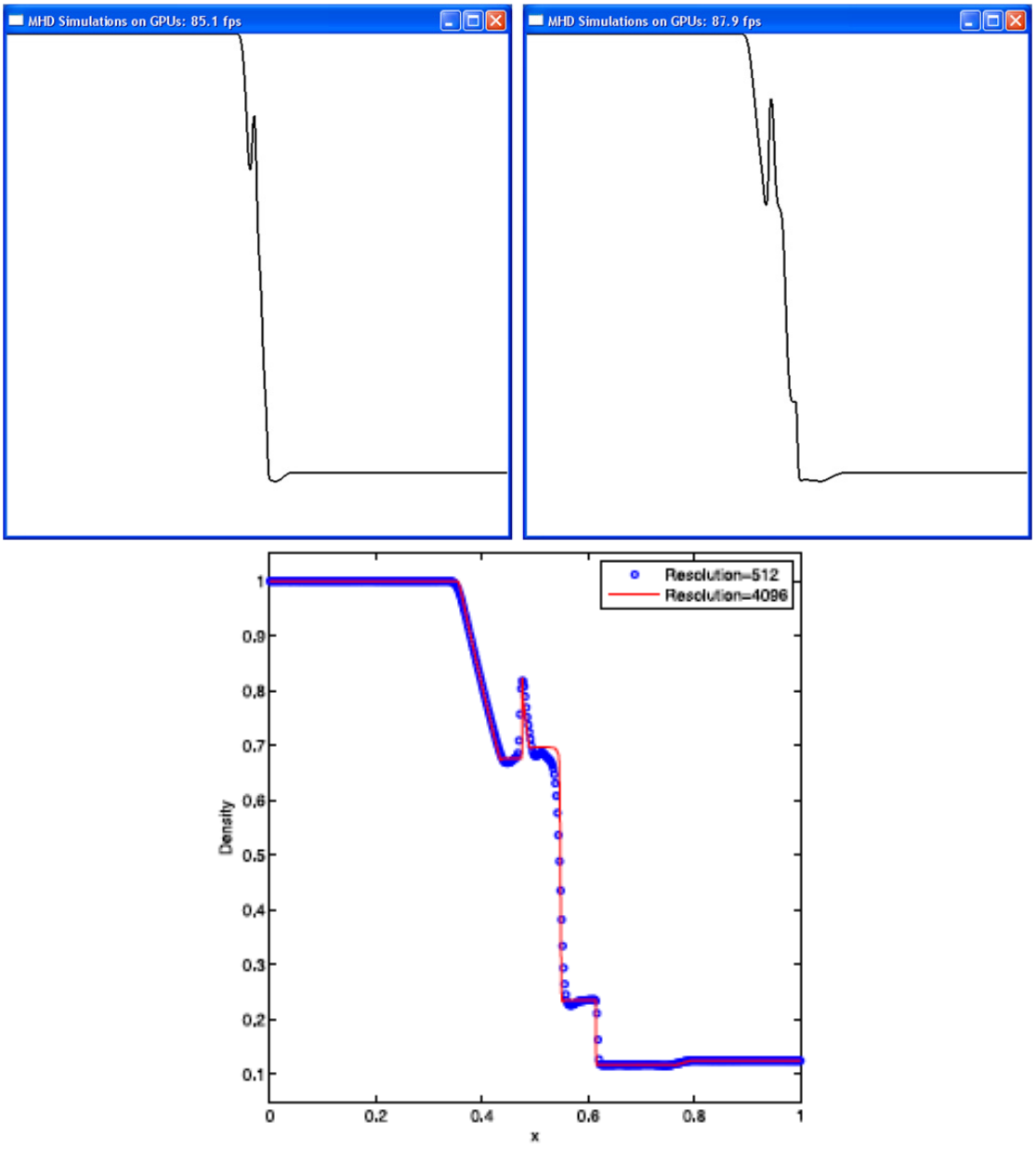

FIGURE 2.4: 1D real-time visualization during the evolution (top) and the corresponding final result (bottom) of the density (ρ) of Brio-Wu shock tube problem with 512 grid points using *GPU-MHD*.

the simulation results are stored in GRAM, so the copying and mapping operations are also fast and do not affect too much to the performance. Although the number of grid points in 2D problem is much larger than those in the one-dimension problem, the FPS still reaches 10 in single precision mode and 6 in double precision mode, still giving acceptable performance to the user. Visualization of 3D problem is different to 1D and 2D problems. GPU-based volume visualization method [68] and texture memory (or video memory) are used. The screen captured images of the real-time visualization are shown in Figures 2.4 to 2.6.

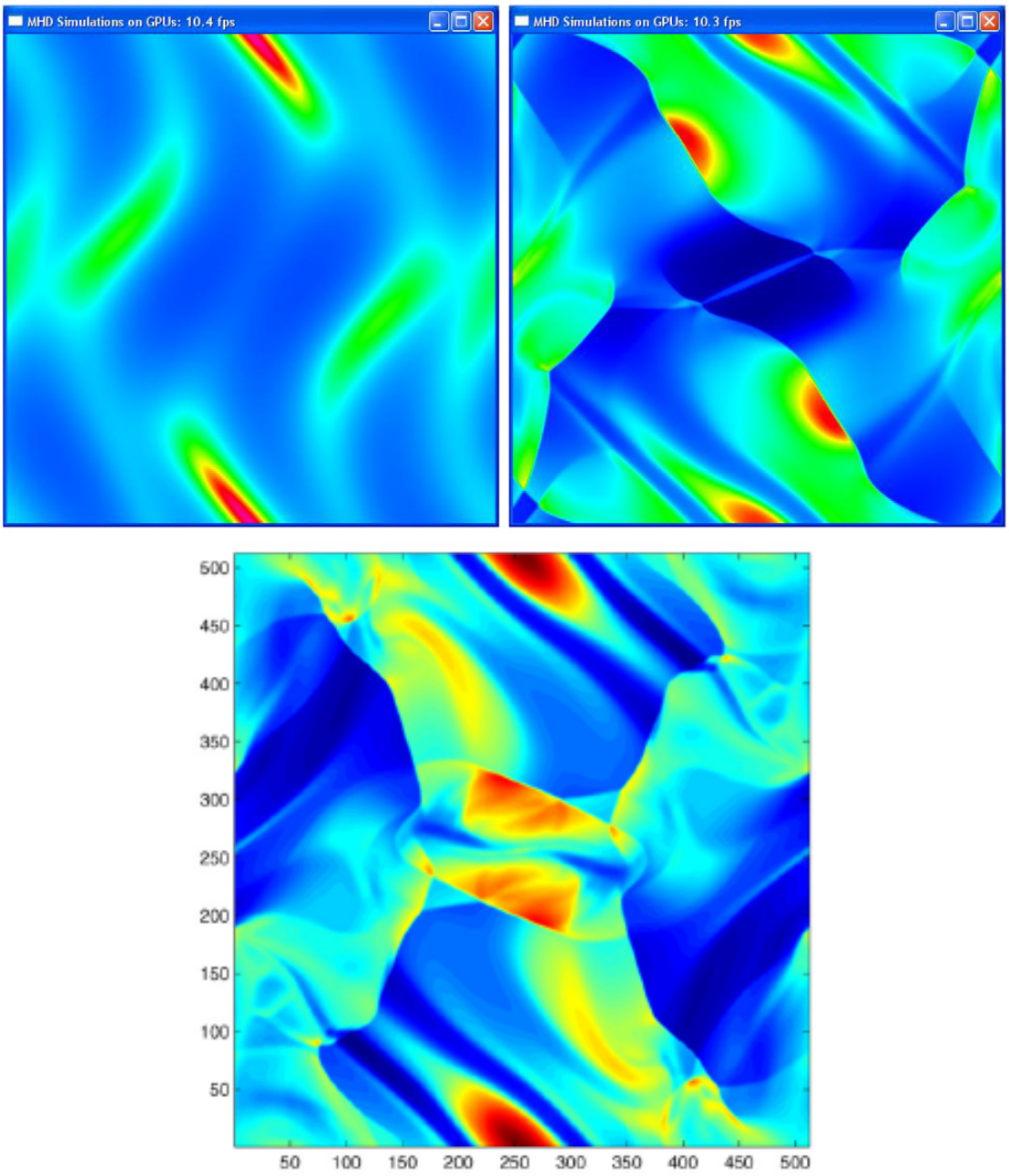

FIGURE 2.5: 2D real-time visualization during the evolution (top) and the corresponding result at $t = 5$s (bottom) of the density (ρ) of Orszag-Tang vortex test with 512^2 grid points using *GPU-MHD*.

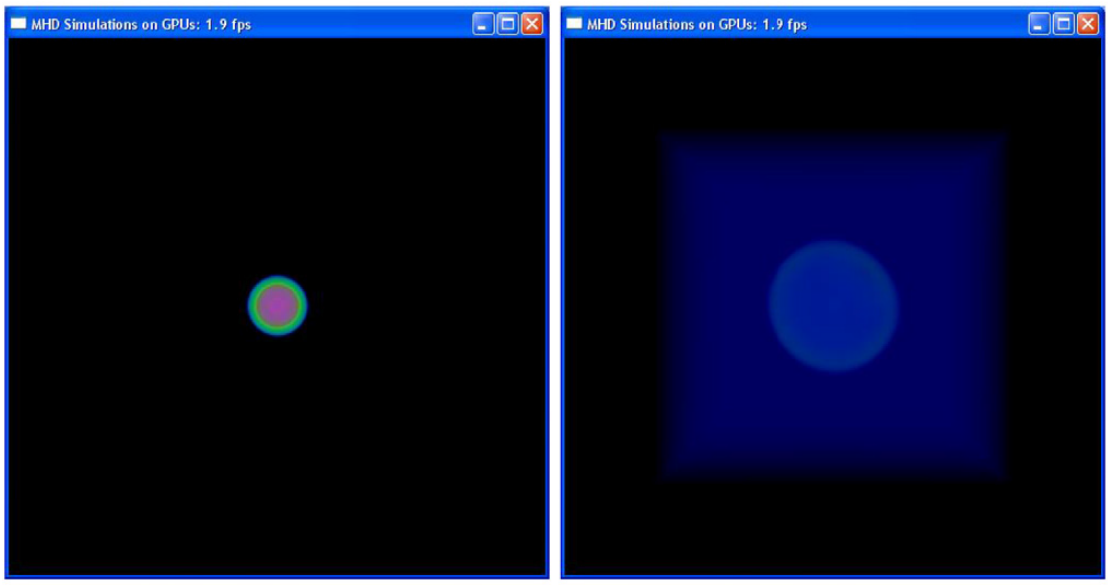

FIGURE 2.6: 3D real-time visualization of the Energy (E) of 3D Blast wave problem with 128^3 grid points using *GPU-MHD*.

The *GPU-MHD* code achieves speedups (in single precision) of about 10× (1D problems with 4096 grid points), 200× (2D problems with 1024^2 grid points), and 84× (3D problems with 128^3 grid points), respectively, compared to the corresponding serial CPU MHD implementation. More implementation details, numerical tests as well as performance measurements and accuracy analysis are given in [57].

Chapter 3

GPU Direct-MPI Hybrid Framework

3.1 Multi-GPU Systems and the Data Communication Overhead

To use GPUs for computations, data have to be stored in the video memory (GRAM) which is on the graphics board. But the capacity of GRAM on a single GPU is far less than the requirement for large-scale simulations. Thus, distributed multi-GPU systems are needed. Nowadays, many supercomputers with high rankings in the Top 500 list [47] use GPUs as the accelerator to offer extremely high computation power, including Titan, Cray CS-Storm, TSUBAME 2.5, and Tianhe-1A. Many-core GPUs provide very high floating point operations per second (FLOPS) and their power are increasing rapidly. The wide use of GPUs also drives their development, where the architecture of GPUs is now being renewed in every two years. For example, the Kepler GPUs have about two times (in double precision) the peak FLOPS [69] of the Fermi GPUs [70], but the bandwidth is not increasing as fast as the computational power. Moreover, data exchanges between GPUs have to go through the host memory (memory on the motherboard). This can cause extra overheads for computations using distributed multi-GPU systems. If the data are fragmented in the GRAM, the overheads will be quite heavy. That's why the common way of using distributed multi-GPU systems for large-scale scientific simulations is to decompose the whole computational domain in two-dimension (2D) (only do the partitioning along the y and z dimensions — where most of the data to be exchanged stored in continues memory address). On the other hand, data transfers between GPUs on distributed multi-GPU systems are also the bottleneck. Because the whole computational domain is decomposed into a certain

number of partitions, and these partitions are stored and calculated in different nodes, it is necessary to exchange the halo data of a partition with its adjacent partitions. Data communication between computing nodes are processed through the network. The supercomputer or cluster typically has highly optimized network specification (such as InfiniBand) and settings for fast data communication. However, data stored in GRAM has to be copied to the main memory before being transferred to another node. It is also necessary to copy the received data from the main memory to the GRAM.

We hereby propose an efficient GPU Direct-MPI hybrid framework, providing fast data communication to tackle the data transfer bottleneck. GPU Direct 2.0 [2, 50] for peer-to-peer data transfer between GPUs has been used in our approach. Our framework can be used for any grid-based numerical simulation including CFD and MHD. An implementation of magnetohydrodynamic (MHD) simulation by the GPU parallelization of a TVD algorithm for solving the multidimensional ideal MHD equations. An extension our previous work *GPU-MHD* code [57] for single GPU computation to multiple GPUs is presented (called *MGPU-MHD* code). The *MGPU-MHD* code has been implemented and tested on the TSUBAME supercomputer [3, 71] at the Tokyo Institute of Technology.

3.2 Efficient Data Communication on Distributed Multi-GPU Systems

Numerical simulations using GPUs usually can achieve high efficiency, though the limited size of the memory on a graphics cards restricts the resolution of a MHD simulation that runs on a single GPU. According to the ideal MHD equations, there are eight physics quantities (ρ, ρv_x, ρv_y, ρv_z, E, B_x, B_y, B_z) for the 3D simulation. Extra memory for storing the intermediate results of the numerical method is also required. In our previous experiments, the maximum resolution of a MHD simulation that runs on a NVIDIA Tesla M2050 GPU with 3GB memory is 225^3. Therefore, distributed multi-GPU systems are needed for large-scale MHD simulations.

In a MHD simulation, the physics quantities of a grid point are updated by using one or more physics quantities of its adjacent grid points. Such calculation is called the stencil calculation. When performing simulations on clusters, the whole calculation domain is divided into many partitions and each partition is stored in a node. When calculating the elements on the boundary of a partition, boundary elements from the adjacent partitions are needed as the inputs for the stencil calculation. Hence, a node needs a copy of the boundary elements of its adjacent partitions in every direction. The copy of the required elements of the adjacent partitions are called the halo (See Figure 3.1).

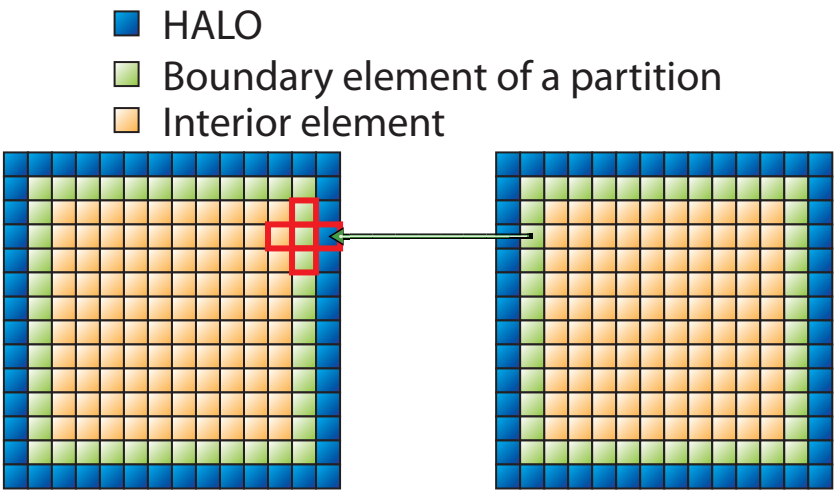

FIGURE 3.1: The halos for the stencil calculation.

The halo is updated at every calculation step by copying the boundary data from the adjacent partitions. Therefore, data exchanges between GPUs and nodes are required, leading to the overhead in every step of the calculation.

For a 3D MHD simulation, there are eight physics quantities and the relaxing TVD scheme for the MHD equations needs four halos in every direction. This huge amount of the halo data exacerbates the data transfer bottleneck. In this section, we introduce several techniques employed in our implementation in order to enhance the efficiency of the MHD simulations on distributed multi-GPU systems.

3.2.1 Overlapping

Overlapping the calculation and data communication is an effective technique used in GPU computing. Similar to the direct memory access (DMA) mechanism of the host system, NVIDIA GPUs with compute capability higher than version 1.1 contain dedicated DMA engines for data transfers over PCIe that work concurrently with CUDA kernel executions. The concept of overlapping for grid-based MHD simulations is to calculate the boundary elements first. Then, renew the halo by copying the updated boundary elements between every partition and calculating the interior grid points simultaneously (see Figure 3.2).

Overlapping has some overhead because of breaking the computation into two steps. The processing time of a kernel is usually $T_{Non-Overlapped} < T_{Overlapped}$. Nevertheless, overlapping can hide the latency of I/O, resulting in a performance gain in total. The

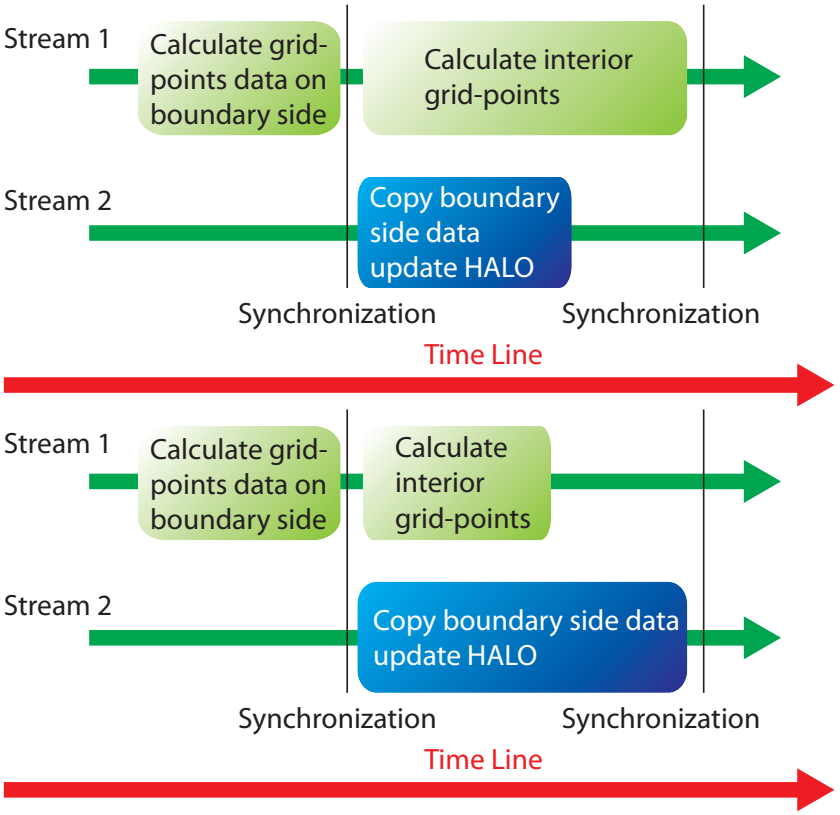

FIGURE 3.2: The concept of overlapping: The two situations of the time expense of Equation 3.3.

time expense can be calculated by the following equations:

$$T_{Non-Overlapped} < T_{Overlapped} = T_{Boundary} + T_{Interior} \tag{3.1}$$

Without overlapping:

$$T_{total} = T_{Non-Overlapped} + T_{DataExchange} \tag{3.2}$$

Overlapping:

$$T_{total} = T_{Overlapped} = T_{Boundary} + \mathbf{max}(T_{Interior}, T_{DataExchange}) \tag{3.3}$$

It will be an ideal situation if the data exchange can be totally hidden by the calculation of the interior grid points (Equation 3.3 where $T_{DataExchange} \leq T_{Interior}$). This situation can happen in GPU applications on a single workstation with multiple GPUs. However, the time for data exchanges between nodes on a cluster is generally longer then the calculation time, and it is even longer for distributed multi-GPU systems. The data

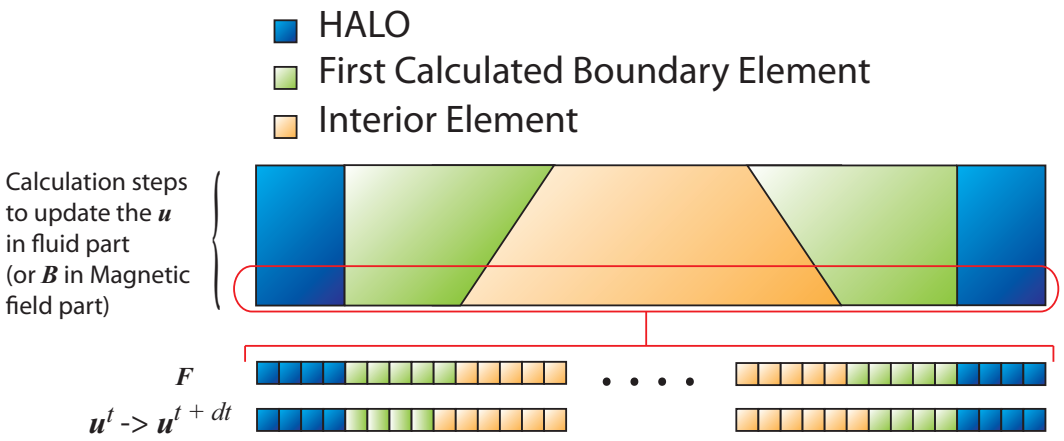

FIGURE 3.3: The number of the pre-computed boundary elements.

of the simulation is stored in the dedicated memory on the graphics board, so a GPU-CPU-GPU copy is required. That means the data has to be copied from the "device" (GPU) memory to the "host" (CPU) memory in a node (so-called "D2H copy"), and then transfer to another node, and copy from the "host" memory to the "device" memory (so-called "H2D copy") of the another. In addition, the amount of the exchanging data is large in MHD simulation.

If the calculation of the interior is hidden by the data transfer, the larger the $T_{Interior}$ compared to the $T_{Boundary}$, better performance will be obtained. The "thickness" of the boundary elements is the same as the size of the halos (= 4 in the TVD MHD method). However, the evolution of the physics quantities of the numerical simulation in a dt requires many calculation steps. And the stencil calculation of each step needs one or more adjacent grid points invoked. In our implementation, we need four boundary elements in each direction to be updated before the overlapping process of the data transfer and the calculation of interior. To update four fluid vectors $\boldsymbol{u}$ of the MHD equations, $4 + 1 = 5$ fluxes $\boldsymbol{F}$ are needed to be calculated first. To get five fluxes, more intermediate results calculated by the boundary elements are required. Even though we only need four updated u. The number of the boundary elements of the intermediate results of each step is an incremental number counting from the last process block to the first process block of Figure 3.2. For both the fluid and the magnetic field parts of every L_i $(i = x, y, z)$, the concept is illustrated in Figure 3.3. The total number of computed elements is the same as the non-overlapped approach. However, the increasing number of the pre-computed boundary elements affects the performance gain of the overlapping technique.

L_x, L_y and L_z of *GPU-MHD* contains the same calculation kernels with different indexing operations (see Figure 2.3 of Section 2.2). To extend the computation of L_i

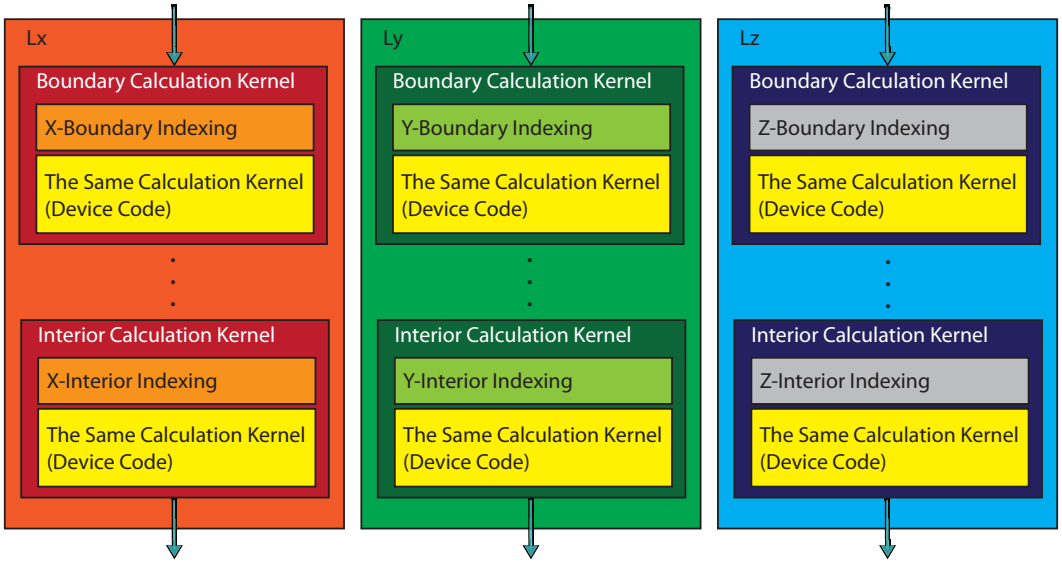

FIGURE 3.4: Computation kernels in *MGPU-TVD* for overlapping.

with overlapping, we only need new index operations for the traversal of the boundary elements and the interior elements. The calculation parts are still the same for both boundary and interior calculations of L_x, L_y, and L_z, as shown in Figure 3.4.

Overlapping somehow enhances the efficiency, but due to the high workload of data exchanges, a better method is still needed to speedup the data exchanges.

3.2.2 GPU Direct-MPI Hybrid Communication

Distributed multi-GPU systems usually have more than one GPU installed on the PCIe interfaces in each node. Before GPU Direct 2.0 was available, data exchanges between two GPUs had to proceed through a GPU-CPU-GPU memory copy. This indirect way dramatically reduces the performance of distributed multi-GPU systems. The overhead is large and cannot be hidden by the overlapping technique. In order to enhance the performance of large-scale MHD simulations on distributed multi-GPU systems, we hereby propose a GPU Direct-MPI hybrid communication approach. In this approach, MPI is only used for the data exchange between computer nodes where GPU Direct 2.0 is applied for the internal data exchange between GPUs within a node. It is different from the commonly used flat MPI approach which launches the same number of processes as the number of GPUs, we only need to launch one process in each node to drive all the GPUs in a node. Therefore, the computational domain is divided by the number of nodes rather than the total number of GPUs of the system. As a result, only two MPI data transfers on each direction are needed per node, as showed in Figure 3.5. Our approach reduces the MPI traffic and achieves higher efficiency in data exchanges,

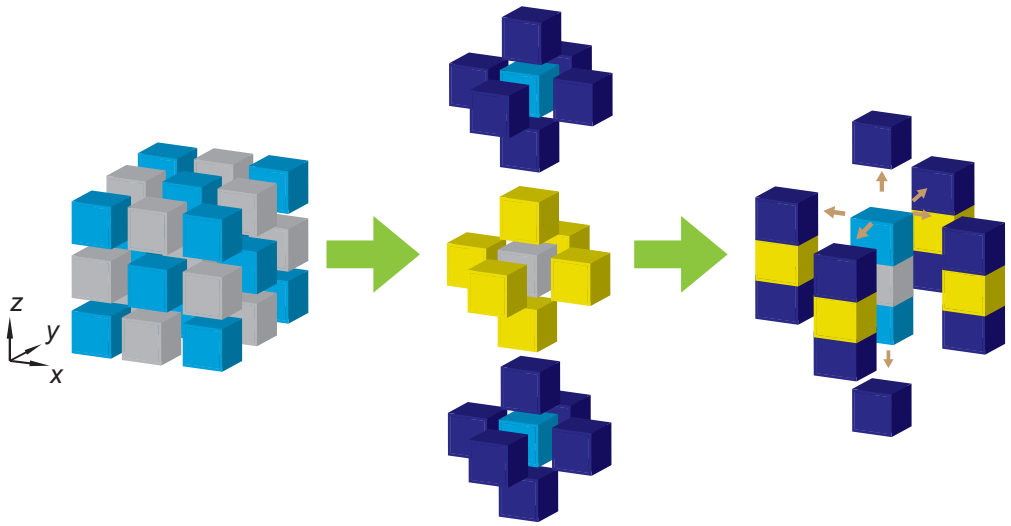

FIGURE 3.5: Multiple partitions in the z-dimension can be handled by multiple GPUs in one single process.

resulting in total efficiency enhancement since the data exchange is the bottleneck of large-scale MHD simulations on distributed multi-GPU systems. More details of the strategies used in our approach:

1. Using GPU Direct for fast internal data exchange between GPUs
2. Using CUDA kernel to speedup the fragmented data exchange,

will be explained in the following subsections.

3.2.2.1 GPU Peer-to-peer Data Transfers

GPU Direct 2.0 is a feature of Fermi or newer generation GPUs and it is available for CUDA 4.0 (since 2011) or later. It provides peer-to-peer data communications between two GPUs in the same node (see Figure 3.6). By using GPU Direct 2.0, data transfers between two GPUs become straightforward and much more efficient. On the other hand, a single process can handle all the GPUs in a node in a easier way by using CUDA runtime APIs. Figure 3.7 shows the difference between two approaches of using flat MPI and GPU Direct 2.0 for data communications between GPUs.

Because multiple GPUs within a node are handled by a single process, the decomposition of the computational domain becomes a two-layer process. At the first layer, the whole computational domain is decomposed by the number of computation nodes of the system, and communication is done using MPI, where one process is assigned to every node. At the second layer, a partition domain in a node is decomposed in one dimension

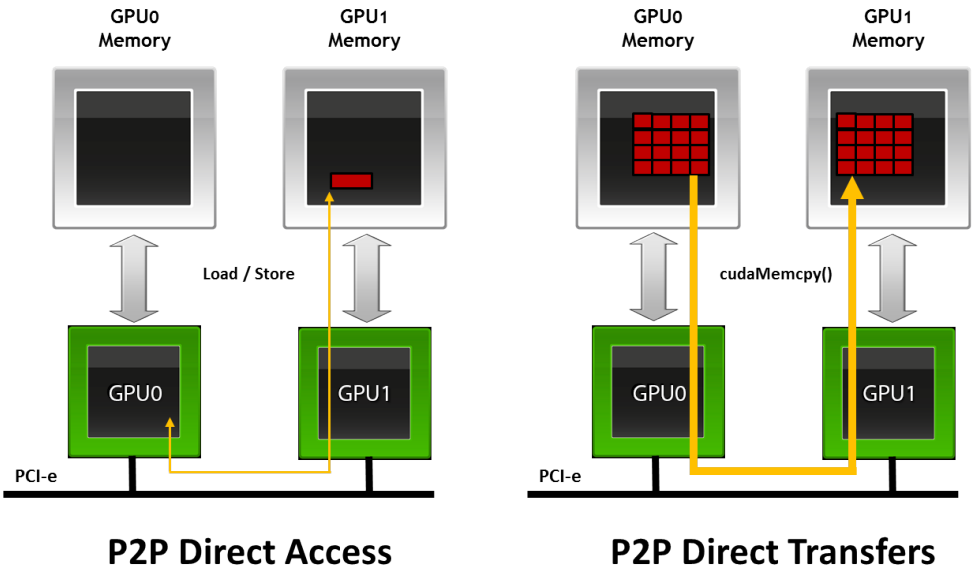

FIGURE 3.6: Peer-to-peer communication between GPUs on the same PCIe bus[2].

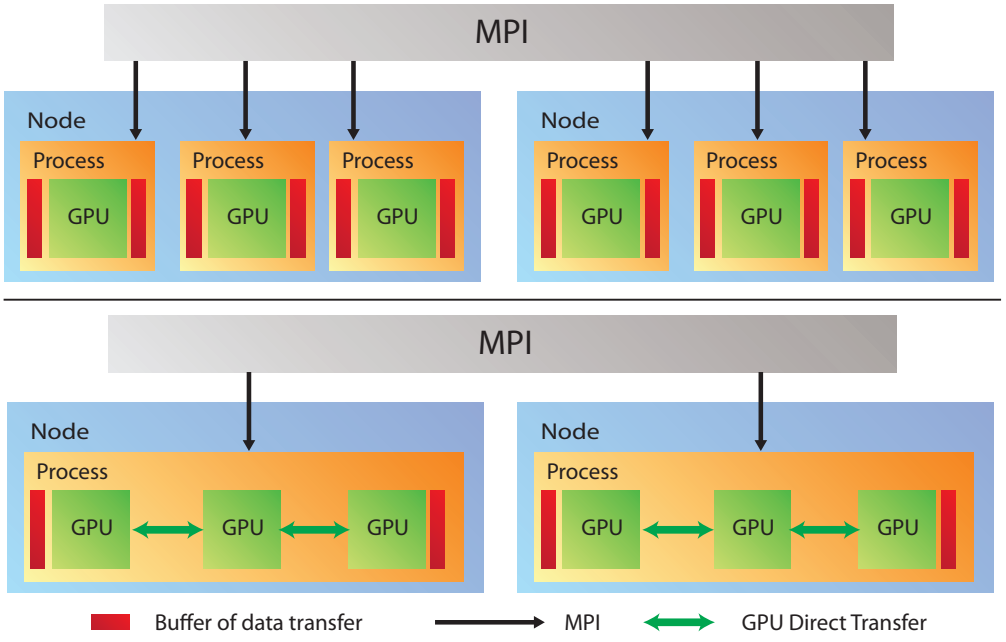

FIGURE 3.7: Comparison between using flat MPI (top) and GPU Direct 2.0 (bottom) for inter-node data exchanges.

(z-dimension in our case) and assigned to every GPU. For the data exchange in the z-direction, only the first GPU and last GPU have to exchange the data to the adjacent node via MPI (see Figure 3.5). The data exchanges in y and x directions can be done by packing the data to a single buffer and transferring the packed data to other nodes via MPI. When a node receives the packed data, it unpacks the data and then distributes them to every GPU in that node. More detail about the data exchange in y and x directions will be explained in the following subsection. The data exchange between GPUs in the z-direction in the same node is done via GPU peer-to-peer data transfer. The `cudaMemcpyAsync` CUDA API is used to exchange the data inside a node instead of `MPI_isend` and `MPI_irecv` (or `MPI_sendrecv`). This approach not only increases the efficiency of the data exchange between the GPUs inside a node, but also reduces the number of MPI communications. Asynchronous copy (non-blocking copy with synchronization) is used to let the GPUs transfer the data concurrently. Furthermore, internal data exchanges using GPU Direct 2.0 transferring data between GPUs via PCIe interface directly. Thus, it can be run simultaneously with MPI transfers. Experimental results shows the great benefit of our approach compared to the common flat MPI approach.

3.2.2.2 Fragmented Data Movement via CUDA Kernel

To run large-scale MHD simulations using many nodes on a distributed multi-GPU system, it is necessary to decompose the whole computational domain into a certain number of partitions and assign each partition to a node for computation. For 3D MHD simulations, in theory, the best way is to use 3D decomposition since each partition will have the smallest surface area. In other words, each partition will have less halo elements which means less memory usage for the halo and less data to be transferred. Thus, it shortens the data exchange time. Besides, since the memory usage of the halo is reduced, the saved memory can be used for the non-halo data. More grid points can be calculated in each node. Table 3.1 shows an example of the halos of a 3D MHD simulation with the resolution = 1080^3 using different decomposition methods. There are eight quantities and four halos in each direction. The calculation of the memory usage of a node is simply by the following equations:

$$
\begin{aligned}
PartitionSize_{withoutHalo} = \quad & PartitionSize_x \\
& \times PartitionSize_y \\
& \times PartitionSize_z \qquad (3.4) \\
PartitionSize_{withHalo} = \quad & (PartitionSize_x + 2 \times Halo) \\
& \times (PartitionSize_y + 2 \times Halo) \\
& \times (PartitionSize_z + 2 \times Halo) \\
= \quad & (PartitionSize_x + 8) \\
& \times (PartitionSize_y + 8) \\
& \times (PartitionSize_z + 8) \qquad (3.5) \\
NumberOfHalo = \quad & NumberOfQuantities \\
& \times (PartitionSize_{withHalo} \\
& - PartitionSize_{withoutHalo}) \\
= \quad & 8 \times (PartitionSize_{withHalo} \\
& - PartitionSize_{withoutHalo}) \qquad (3.6) \\
MemoryUsage = \quad & (NumberOfHalo) \times (BytesOfVariable) \\
= \quad & (NumberOfHalo) \times \tfrac{8}{1024 \times 1024 \times 1024} \qquad (3.7)
\end{aligned}
$$

Where $PartitionSize$ is the size of a partition domain. For example, $PartitionSize$ of the 1080^3 domain with $6 \times 6 \times 6$ decomposition is :
$(PartitionSize_x, PartitionSize_y, PartitionSize_z) = (1080/6, 1080/6, 1080/6) = (180, 180, 180)$.

Unfortunately, the I/O of the fragmented data of the device memory has extremely low performance (see Figure 3.8). In our experience, it spends dozens or even hundreds of times for copying the halo data in x-direction compared to z-direction. However, we found that moving the data via a CUDA kernel (simply launch a "$a = b$" CUDA kernel) preforms a fast data copy. This improves the efficiency of the data transfer in the 3D decomposition. As a result, 3D decomposition becomes useful and is used in our large-scale MHD simulation implementation. Moreover, it shows higher efficiency than the commonly used 2D decomposition approach.

The procedures of the data exchange in x and y-directions are as follows and and shown in Figure 3.9.

1. For each GPU, copy the data beside the halo to a continue/linear memory space of the same GPU.

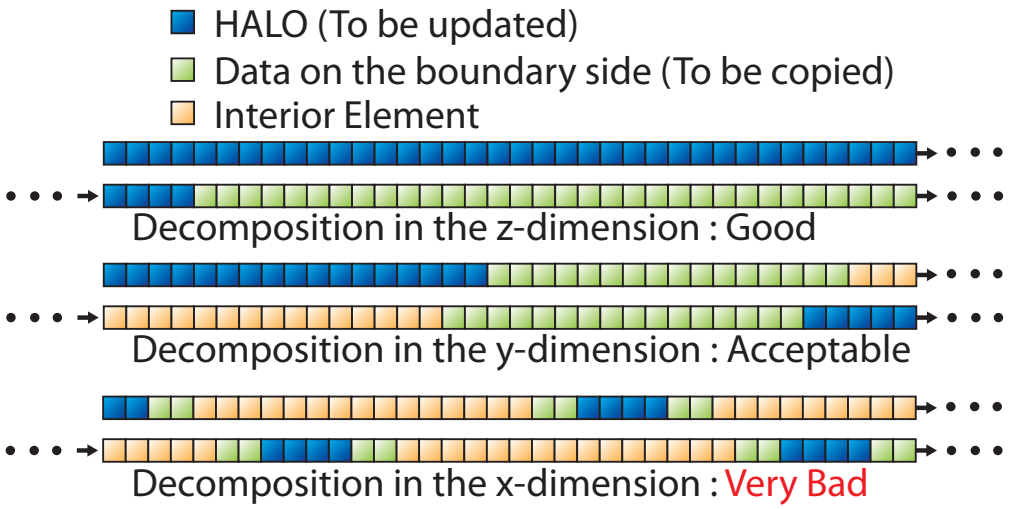

FIGURE 3.8: Data addresses on the memory: Boundary data in the x-boundary is fragmented and shows poor performance in memory copy.

TABLE 3.1: The number of HALO elements and its memory usage (double precision) in different decomposition methods of a 1080^3 MHD simulation.

Decomposition	Number of the halo per node	Memory usage (GBytes)
$1 \times 1 \times 216$	76453376	0.56962204
$1 \times 12 \times 18$	11347456	0.084545135
$6 \times 6 \times 6$	6501376	0.048439026

2. All GPUs copy the data to one single host buffer via asynchronous copy.

3. Copy host memory of the combined data to adjacent node via MPI.

4. Copy the data from the host buffer to a continuous/linear memory space of each GPU via asynchronous copy.

5. For each GPU, copy the data to the halo from the continuous/linear memory space of the same GPU.

Copying the fragmented data directly from device to host is very slow. A buffer is used to align the data before the D2H and H2D copy. GPU has high bandwidth in device-to-device (D2D) copy. Copying data to a linear buffer and then proceeding the D2H or H2D copy shows better performance than a D2H/H2D copy directly with the discontinued memory space of the device memory.

In Steps 1 and 5, D2D copy is done within each GPU concurrently. For the D2D copy in each GPU, CUDA asynchronous copy is used in the y-direction where as a CUDA kernel is used in the x-direction. As we mentioned, data copy of the fragmented data in the x-direction is very slow but it is fast if we use a CUDA kernel. In our experimental results, the average transfer rate of a whole process of the data copy including all five steps we

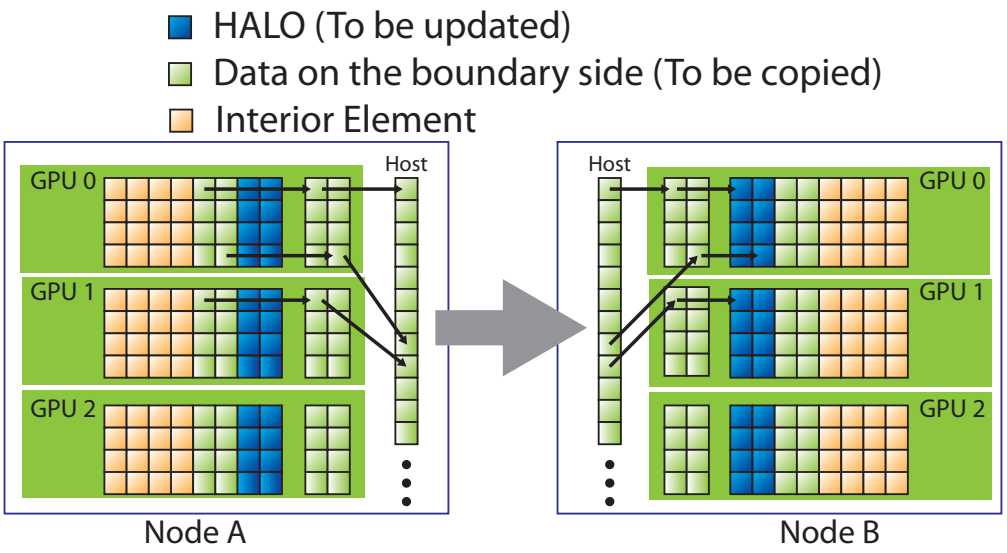

FIGURE 3.9: Data exchange in x and y directions. Data copy between device memory of each GPU and host memory are concurrent.

listed above was 0.153 GB/s in the y-direction and 0.174 GB/s in the x-direction. The data exchange in x-direction can even provide faster speed compared to the y-direction in a breakdown measurement. However, since the CUDA kernel is launched for the data exchange in x-direction, overlapping technique cannot be applied for the data exchange in x-direction. Therefore, we only use data copy with overlapping for y-direction and z-direction. For x-direction, a non-overlapped CUDA kernel data copy is used.

In Step 2, each GPU copies the data to the buffer of a single host memory with different starting points of the address. The packed data from all GPUs will be transferred to another node via one MPI transfer process. After that, the packed data will be distributed to each GPU on the other node using a D2H copy. CUDA asynchronous copy are used in Steps 2 and 4, allowing multiple GPUs to copy data to/from the host buffer simultaneously.

Using CUDA kernels for fragmented data exchanges instead of `cudaMemcpy` is simple but the benefit is huge. In the common flat MPI approach on distributed multi-GPU systems, 3D decomposition is always much slower than 2D decomposition due to the low efficiency of the fragmented data exchange. The breakthrough of making 3D decomposition available on distributed multi-GPU systems, minimizes the number of halos in large-scale MHD simulations, resulting in a performance gain in both data exchanges and calculations.

3.3 Numerical Tests

The ideal MHD simulation of our previous work using the *GPU-MHD* code [57] is adopted in our implementation for testing the performance of the GPU Direct-MPI Hybrid framework for large – scale MHD simulations. For more detailed information about the numerical scheme, please refer to Section 2.2, Appendix A and [57].

In this section, we present several 2D and 3D numerical tests for validating our implementation. All tests were performed in double precision on the TSUBAME 2.0 supercomputer at the Tokyo Institute of Technology.

3.3.1 MHD Rotor Problem

The first test is the 2D MHD rotor problem taken from [4]. It initiates a high density rotating disk with radius $r_0 = 0.1$ of fluid measured from the center point $(x, y) = (0.5, 0.5)$. The ambient fluid outside of the spherical region of $r_1 = 0.115$ has low density and $v_x = v_y = 0$. The fluid between the high density disk fluid and ambient fluid ($r_1 > r > r_0$, where $r = \sqrt{(x-0.5)^2 + (y-0.5)^2}$) has linear density and angular speed profile with $\rho = 1 + 9f, v_x = -fv_0(y-0.5)/r$ and $v_y = fv_0(x-0.5)/r$ where $f = (r_1 - r)/(r_1 - r_0)$. The initial condition is listed as follows

$r < r_0$

$$\begin{Bmatrix} v_x \\ v_y \\ v_z \end{Bmatrix} = \begin{Bmatrix} -v_0(y-0.5)/r_0 \\ v_0(x-0.5)/r_0 \\ 0 \end{Bmatrix} \tag{3.8}$$

$r_0 < r < r_1$

$$\begin{Bmatrix} v_x \\ v_y \\ v_z \end{Bmatrix} = \begin{Bmatrix} -fv_0(y-0.5)/r \\ fv_0(x-0.5)/r \\ 0 \end{Bmatrix} \tag{3.9}$$

$r > r_1$

$$\begin{Bmatrix} v_x \\ v_y \\ v_z \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \tag{3.10}$$

$$\rho = \begin{cases} 10 & r < r_0 \\ 1 + 9f & r_0 < r < r_1 \\ 1 & r > r_1 \end{cases} \tag{3.11}$$

$$\begin{aligned} &\text{spherical region center} = (0.5, 0.5) \\ &r_0 = 0.1,\ r_1 = 0.115 \\ &f = (r_1 - r)/(r_1 - r_0),\ (0 \le x \le 1),\ (0 \le y \le 1) \end{aligned} \tag{3.12}$$

A set of initial value of v_0, p, B_x and γ provided in [4] was tested

$$v_0 = 1, \quad p = 0.5, \quad \gamma = 5/3, \quad \left\{ \begin{array}{c} B_x \\ B_y \\ B_z \end{array} \right\} = \left\{ \begin{array}{c} 2.5/\sqrt{4\pi} \\ 0 \\ 0 \end{array} \right\} \tag{3.13}$$

Figure 3.10 presents the images of the density computed with 600 $\times$ 600 grid points. The results are in excellent agreement with those presented in [4].

3.3.2 Blast Wave Problem

The second test is the 2D MHD blast wave problem. The MHD spherical blast wave problem of Zachary *et al.* [72] is initialized by an over pressured region in the center of the domain. The result is a strong outward moving spherical shock with rarified fluid inside the sphere. We followed the test suite [5] of Athena [14]. The initial condition for 2D MHD blast wave problem is listed as follows [5]

$$\left\{ \begin{array}{c} v_x \\ v_y \\ v_z \end{array} \right\} = \left\{ \begin{array}{c} 0 \\ 0 \\ 0 \end{array} \right\} \tag{3.14}$$

$$\left\{ \begin{array}{c} B_x \\ B_y \\ B_z \end{array} \right\} = \left\{ \begin{array}{c} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{array} \right\} \tag{3.15}$$

$$\begin{aligned} &\rho = 1, \quad \gamma = 5/3, \quad p = \left\{ \begin{array}{ll} 10 & \text{inside the spherical region} \\ 0.1 & \text{outside the spherical region} \end{array} \right. \\ &\text{spherical region center} = (0.5, 0.5),\ r = 0.1 \\ &(0 \le x \le 1),\ (0 \le y \le 1) \end{aligned} \tag{3.16}$$

In Figure 3.11, we present images of the density computed with 600 $\times$ 600 grid points. The results are in excellent agreement with those presented in [5].

The third test is the 3D MHD spherical blast wave problem. The velocity, density ρ, pressure p, and γ are the same as those in 2D case. The different part of the initial

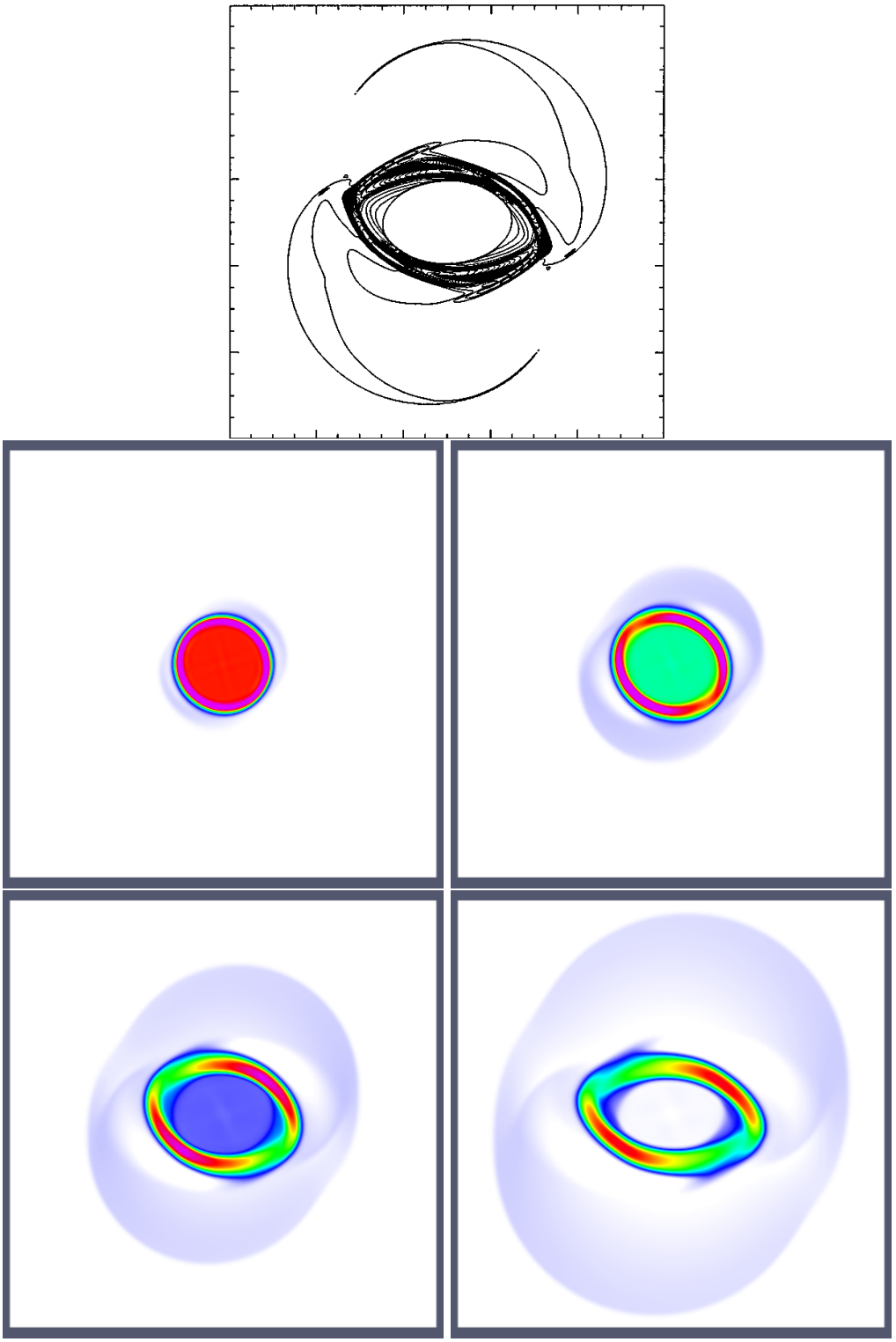

FIGURE 3.10: The figure given in [4] (top, $t = 0.15$s) and our simulation results of the density of the 2D MHD rotor problem computed with 600×600 grid points at $t =$ 0.01s (middle-left), 0.09s (middle-right), 0.17s (bottom-left), and 0.25s (bottom-right).

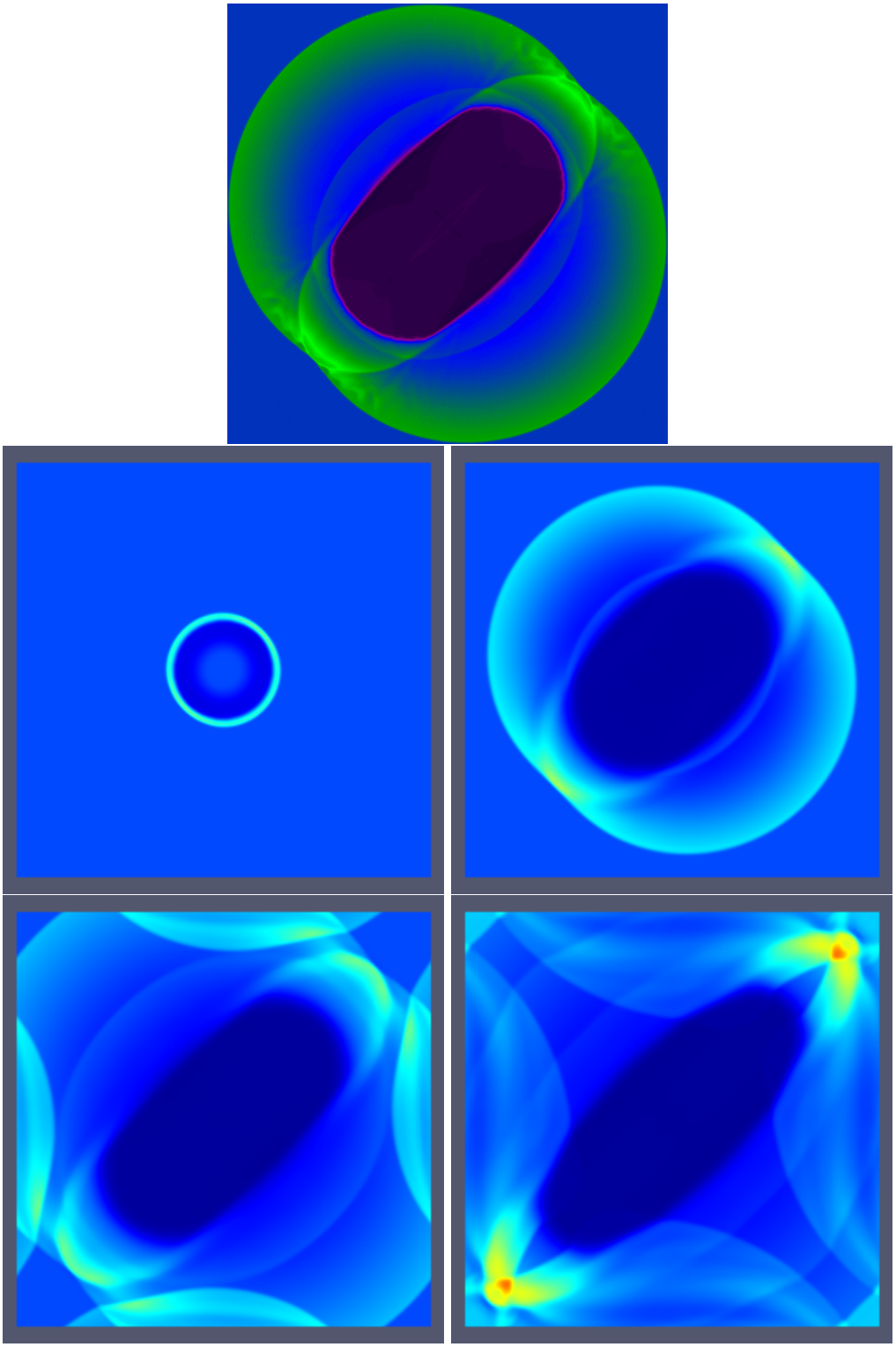

FIGURE 3.11: The figure given in [5] (top, $t = 0.2$s) and our simulation results of the density of the 2D blast wave problem computed with 600×600 grid points at $t = 0.01$s (middle-left), 0.16s (middle-right), 0.29s (bottom-left), and 0.38s (bottom-right).

condition is listed as follows [5]

$$\begin{Bmatrix} B_x \\ B_y \\ B_z \end{Bmatrix} = \begin{Bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{Bmatrix} \tag{3.17}$$

$$\begin{aligned} p &= \begin{cases} 10 & \text{inside the spherical region} \\ 0.1 & \text{outside the spherical region} \end{cases} \\ &\rho = 1, \quad \gamma = 5/3 \\ &\text{spherical region center} = (0.5, 0.5, 0.5), \ r = 0.1 \\ &(0 \le x \le 1), \ (0 \le y \le 1), \ (0 \le z \le 1) \end{aligned} \tag{3.18}$$

Figure 3.12 shows the results of the density computed with 300 × 300 × 300 grid points. Due to the scarcity of published 3D test results, we are not able make direct comparisons with results presented in the literature here.

3.3.3 Orszag-Tang Problem

The fourth test is the 2D Orszag-Tang vortex problem [73], which is used to study incompressible MHD turbulence. In our test, the boundary conditions are periodic everywhere. The density ρ, pressure p, initial velocities (v_x, v_y, v_z), and magnetic field (B_x, B_y, B_z) are given by

$$\begin{Bmatrix} v_x \\ v_y \\ v_z \end{Bmatrix} = \begin{Bmatrix} -\sin(2\pi y) \\ \sin(2\pi x) \\ 0 \end{Bmatrix} \tag{3.19}$$

$$\begin{Bmatrix} B_x \\ B_y \\ B_z \end{Bmatrix} = \begin{Bmatrix} -B_0 \sin(2\pi y) \\ B_0 \sin(4\pi x) \\ 0 \end{Bmatrix} \quad \text{where } B_0 = 1/\sqrt{4\pi} \tag{3.20}$$

$$\rho = 25/(36\pi), \quad p = 5/(12\pi), \quad \gamma = 5/3, \quad (0 \le x \le 1), \ (0 \le y \le 1) \tag{3.21}$$

The 2D Orszag-Tang vortex test was performed in a 2D periodic box with 600 × 600 grid points. The results of the density are shown in Figure 3.13, where the complex pattern of interacting waves is perfectly recovered. The results agree well with those in Lee et al. [6].

The fifth test is the 3D Orszag-Tang vortex obtained from [74], which is used to study the creation and evolution of small-scale structures in 3D MHD flows. In our test, the boundary conditions are periodic everywhere. The density ρ, pressure p, and γ are the

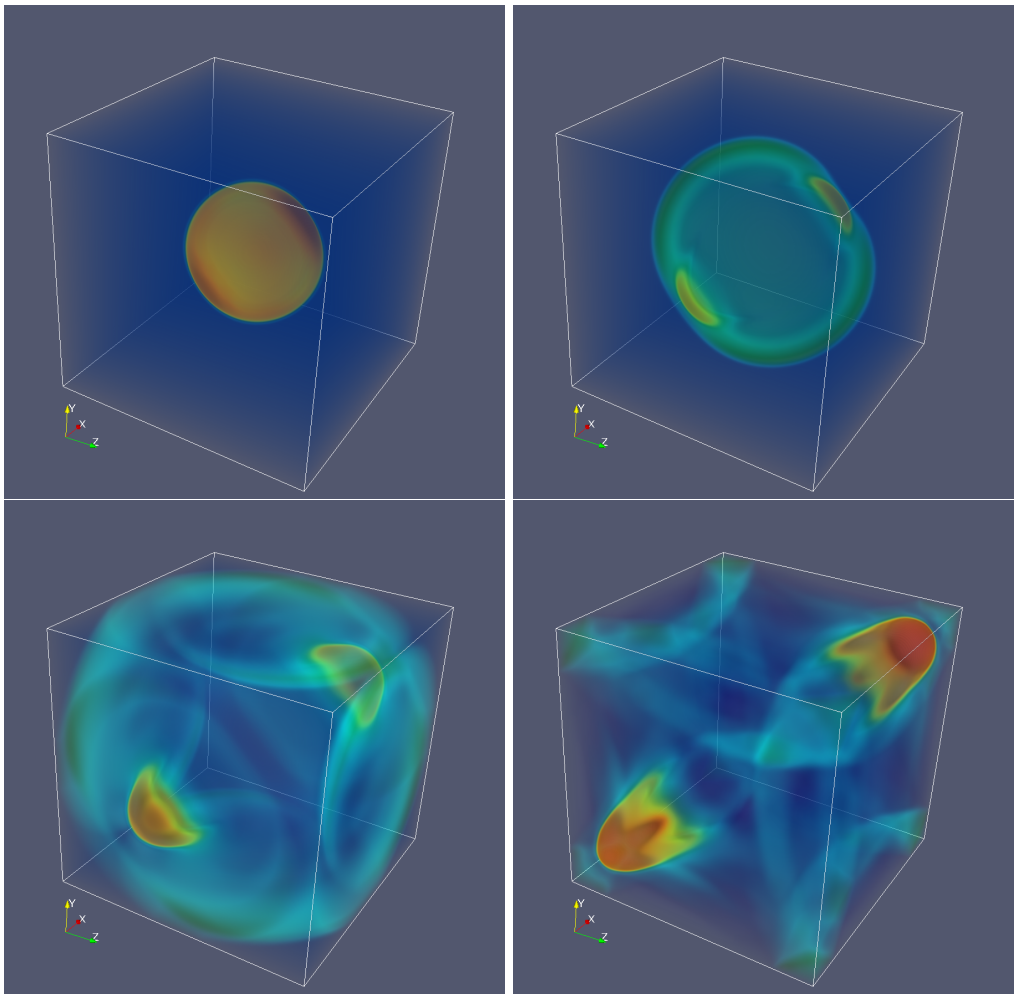

FIGURE 3.12: Results of the density of the 3D blast wave problem computed with 300 × 300 × 300 grid points at t = 0.07s (top-left), 0.18s (top-right), 0.41s (bottom-left), and 0.62s (bottom-right).

same as those in 2D case. Initial velocities (v_x, v_y, v_z), and magnetic field (B_x, B_y, B_z) are given by

$$\begin{Bmatrix} v_x \\ v_y \\ v_z \end{Bmatrix} = \begin{Bmatrix} -2\sin(2\pi y) \\ 2\sin(2\pi x) \\ 0 \end{Bmatrix} \tag{3.22}$$

$$\begin{Bmatrix} B_x \\ B_y \\ B_z \end{Bmatrix} = \begin{Bmatrix} B_0[2\sin(4\pi y) + \sin(2\pi z)] \\ B_0[2\sin(2\pi x) + \sin(2\pi z)] \\ B_0[\sin(2\pi x) + \sin(2\pi y)] \end{Bmatrix} \quad where B_0 = 1/\sqrt{4\pi}. \tag{3.23}$$

$$(0 \le x \le 1),\ (0 \le y \le 1),\ (0 \le z \le 1) \tag{3.24}$$

The 3D Orszag-Tang vortex test was performed in a 3D periodic box with 300 × 300 × 300 grid points. The results of the density are shown in Figure 3.14, where the

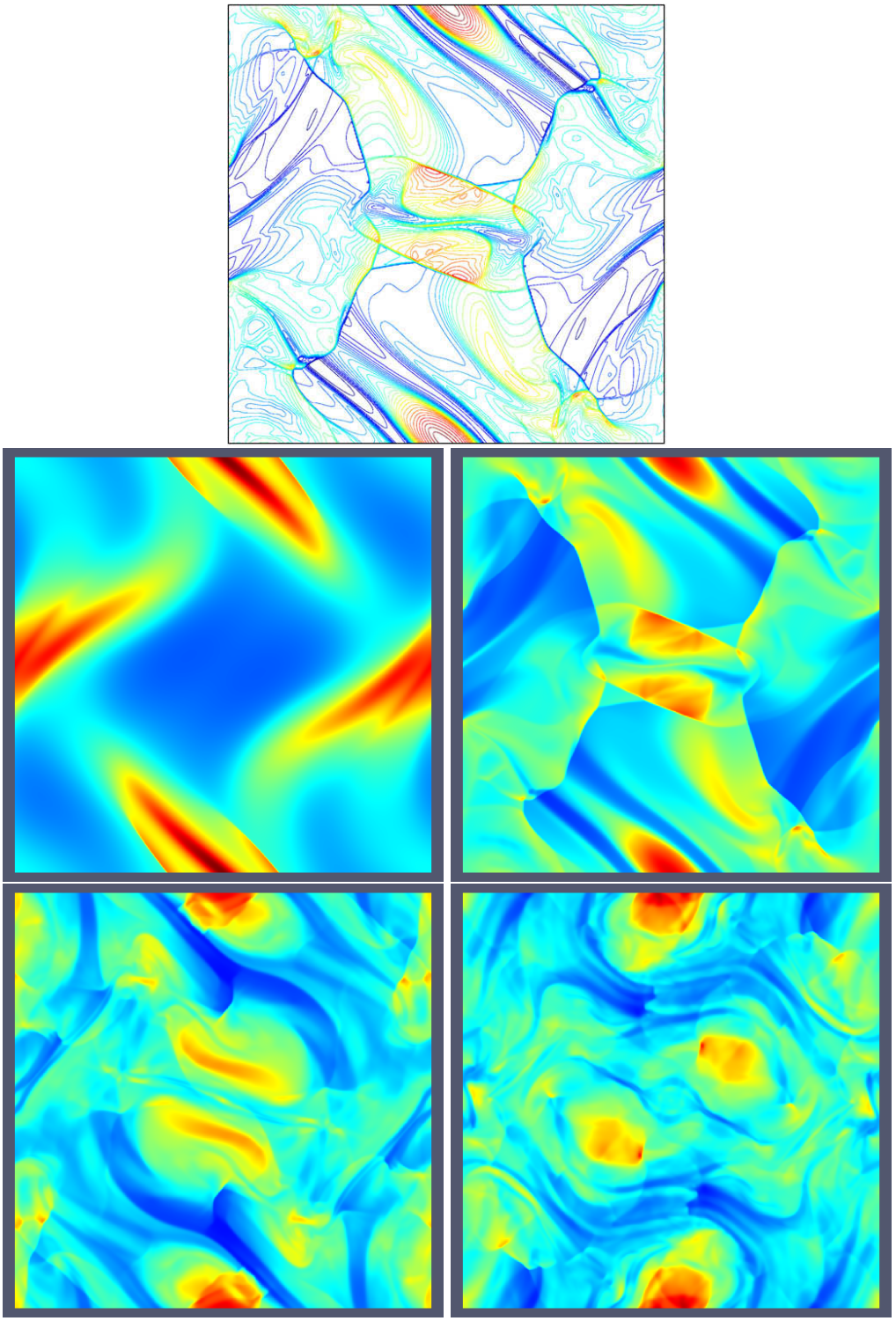

FIGURE 3.13: The figure given in [6] (top, $t = 0.5$s) and our simulation results of the density of the 2D Orszag-Tang vortex problem computed with 600×600 grid points at $t = 0.15$s (middle-left), 0.50s (middle-right), 0.83s (bottom-left), and 1.19s (bottom-right).

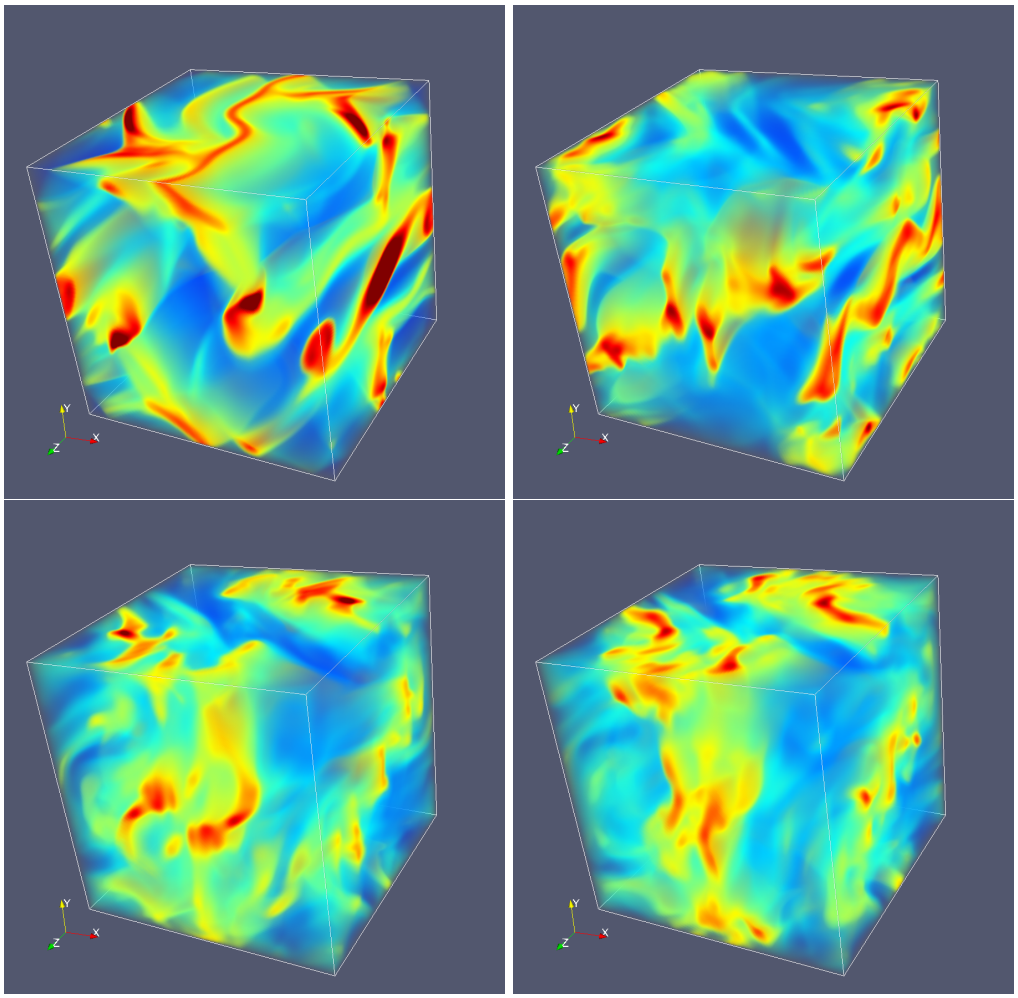

FIGURE 3.14: Results of the density of the 3D Orszag-Tang vortex problem computed with $300 \times 300 \times 300$ grid points at $t = 0.24$s (top-left), 0.35s (top-right), 0.52s (bottom-left), and 0.57s (bottom-right).

complex pattern of interacting waves is perfectly recovered. Unfortunately, there is lack of published 3D test results, so that we are not able make direct comparisons here.

3.4 Performance Measurements

The performance measurements of our code — *MGPU-MHD* are carried out in this section. All the tests were performed in double precision on the TSUBAME 2.0 supercomputer [3, 71] at the Tokyo Institute of Technology using 216 NVIDIA Tesla M2050 GPUs (72 nodes). Before we analyze the performance, we briefly explain the configuration of a calculation node of the TSUBAME 2.0 shown in Figure 1.7 in Section 1.4.2. Each calculation node has three NVIDIA Tesla M2050 (Fermi architecture) GPUs installed and each GPU contains 3GB device memory.

TABLE 3.2: The comparison of different decomposition methods with respect to the elapse time on TSUBAME 2.0 (ms/step).

Decomposition	Flat MPI	GPU Direct-MPI Hybrid	Speedup
$6 \times 6 \times 6$	937.13	776.59	1.2
$1 \times 18 \times 12$	1362.00	1092.80	1.25
$1 \times 12 \times 18$	1299.46	974.05	1.33
$1 \times 1 \times 216$	4708.47	3197.14	1.47

Data transfers between GPU0 and GPU1 or GPU2 cannot be done via peer-to-peer transfer since there are two IOHs between them. CUDA will automatically switch to GPU-CPU-GPU data copy without the requirement of allocating extra host memory. The transfer rate between GPU0 and GPU1 or GPU2 is from 2.88 to 3.56 GB/s and it is 4.9 GB/s using peer-to-peer transfer between GPU1 and GPU2 on a calculation node of the TSUBAME 2.0.

Performance measurements are shown in Table 3.2 and Figure 3.15. The resolution of the whole computational domain was 864^3 in all tests. By using CUDA kernels to move the halo data along the x-direction, simulations using the 3D decomposition achieved about 1.39× faster then those using 2D decomposition flat MPI approach. Furthermore, we obtained about 1.2× speedup more (1.67× in total) by using GPU Direct 2.0. Having the fastest elapse time, simulations using 3D decomposition with GPU Direct 2.0 showed enhancements in performance, supporting the benefits of our proposed method. In addition, 3D decomposition reduced the memory usage for the halo data, thus each GPU could calculate more data. In our tests using 216 GPUs, the maximum resolution of the whole computational domain that could be processed was 1200^3 for 3D decomposition and achieved 2 TFLOPS while 1D decomposition and 2D decomposition could only performed 0.42 TFLOPS and 1.4 TFLOPS, respectively. Without using our GPU Direct-MPI hybrid communication, the common 2D decomposition flat MPI implementation only achieved 1 TFLOPS. In conclusion, we achieved 2× speedup with our 3D decomposition method using GPU Direct-MPI hybrid approach.

Data exchange is a bottleneck preventing the improvement of the performance of numerical simulations on distributed multi-GPU systems. To further analyze our data transfer techniques for performance enhancements, tests and analysis of the data transfer of our GPU Direct-MPI hybrid communication using 216 GPUs were done. For breakdown measurements, we got the data exchange rate per node as 0.174 GB/s, 0.153 GB/s, and 0.462 GB/s along the x-, y-, and z-directions, respectively.

For the data exchanges along the z-direction, only the GPU0 or GPU3 is invoked in the data exchanges between nodes. The amount of data transferred via MPI is about $\frac{1}{3}$ of that of the total data exchange in the x or y direction. Data exchanges between GPUs

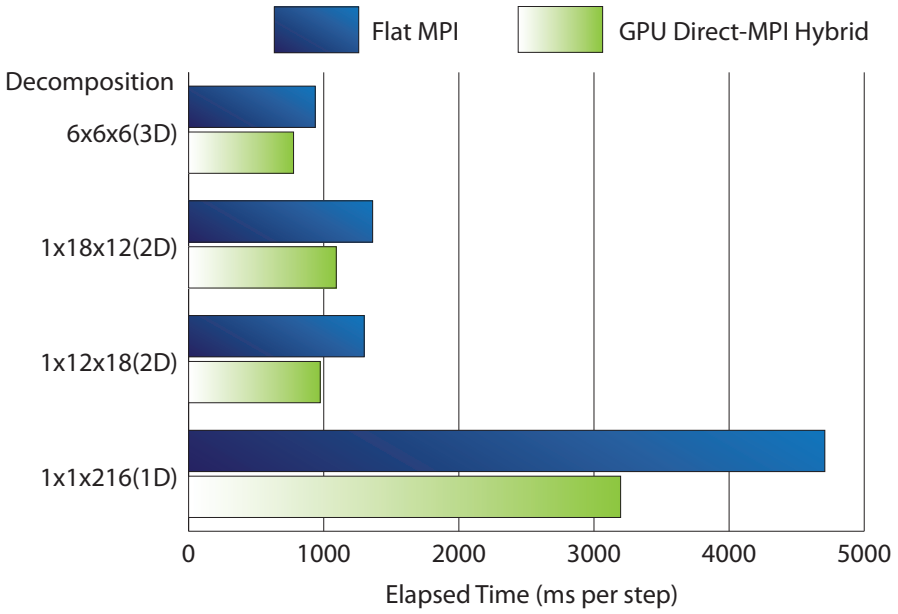

FIGURE 3.15: The comparison of different decomposition methods with respect to the elapse time (ms/step) (Resolution: 864^3).

in the same node are performed using the non-blocking and concurrent peer-to-peer transfers. In addition, data exchanges in different directions contain different overheads:

1. In the x-direction — Fragmented data copy to linear memory via CUDA kernels.
2. In the y-direction — Fragmented data copy to linear memory via CUDA memory copy.
3. In the z-direction — Synchronization of the non-blocking memory between GPUs.

As a result, the data transfer rate of the data exchange along the z-direction is about $3\times$ of that of the data exchange in the other two directions.

Applying overlapping technique results in 11% performance gain. As we mentioned in Section 3.2.2.2, data exchanges along the x-direction block the overlapping. Therefore, overlapping is only applied to the y- and z-directions.

On the other hand, the number of the pre-computed boundary elements increases with respect to the number of the calculation steps (see Figure 3.3 in Section 3.2.1). $T_{Boundary}$ increases when $T_{Interior}$ decreases. Therefore, the overhead ($T_{Interior}$ in this case) can be hidden by the overlapping. We are aware that restricts the performance enhancement when applying overlapping technique to 3D MHD simulations.

Table 3.3 and Figure 3.16 show the results of the data transfer efficiency of the whole data exchange process without the calculation part of our *MGPU-MHD* code. It can be

TABLE 3.3: The comparison of different decomposition methods with respect to the data transfer rate using 216 GPUs on TSUBAME 2.0. (GBytes/s).

Decomposition	Flat MPI	GPU Direct-MPI Hybrid	Ratio
$1 \times 72 \times 3$	6.72558	6.62219	0.98
$1 \times 36 \times 6$	11.1391	12.4836	1.12
$1 \times 18 \times 12$	15.2803	18.9758	1.24
$1 \times 12 \times 18$	15.5393	22.182	1.43
$1 \times 6 \times 36$	9.57825	19.0974	1.99
$1 \times 3 \times 72$	5.64327	10.8108	1.93

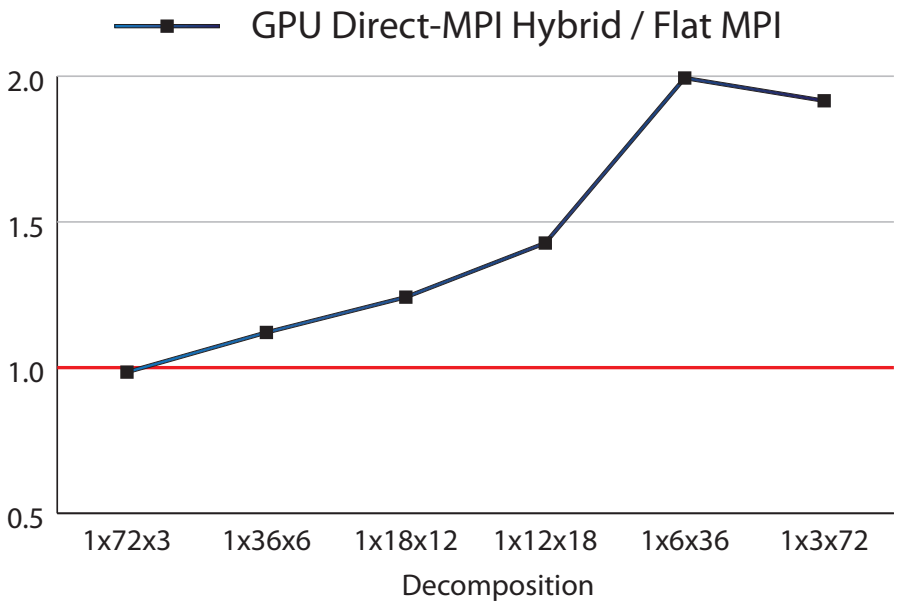

FIGURE 3.16: The ratio of the data transfer rate between flat MPI approach and GPU Direct-MPI hybrid communication using 216 GPUs ($\frac{GPUDirect}{MPI}$).

seen that we obtain a greater benefit of the data exchange along the z-direction. This benefit increases if we have more partitions along the z-direction. The reason is that GPU Direct 2.0 is invoked for in-node GPU data exchanges. The larger number of z-decomposition the higher speedup we get, compared to the flat MPI approach. Besides, the $1 \times 72 \times 3$ decomposition (it is almost a 1D decomposition along the y-dimension) was only a little bit slower because of the overhead of the data exchange along the y-direction. For the data exchanges along the y-direction, both MPI only and GPU Direct approaches copy the data into a linear buffer on the GPU. The overhead was caused by the synchronization of the non-blocking copies. The same overhead also exists in the data exchanges along the x-direction. According to these results, it is evident that the overhead is small and acceptable.

The above performance measurements show that our GPU Direct-MPI hybrid communication enhances the performance in many aspects, achieving great speedups of large-scale MHD simulations on distributed multi-GPU systems. Experimental tests of ideal MHD

TABLE 3.4: Comparison of different decomposition methods with respect to the elapse time (ms / step) on TSUBAME 2.5 (Resolution: 1296^3)

Decomposition	Flat MPI	GPU Direct-MPI Hybrid	Speedup
$6 \times 6 \times 6$	2500.30	2077.36	1.2
$1 \times 18 \times 12$	2967.00	2218.16	1.34
$1 \times 12 \times 18$	2773.12	2044.43	1.36
$1 \times 1 \times 216$	10719.40	5936.13	1.8

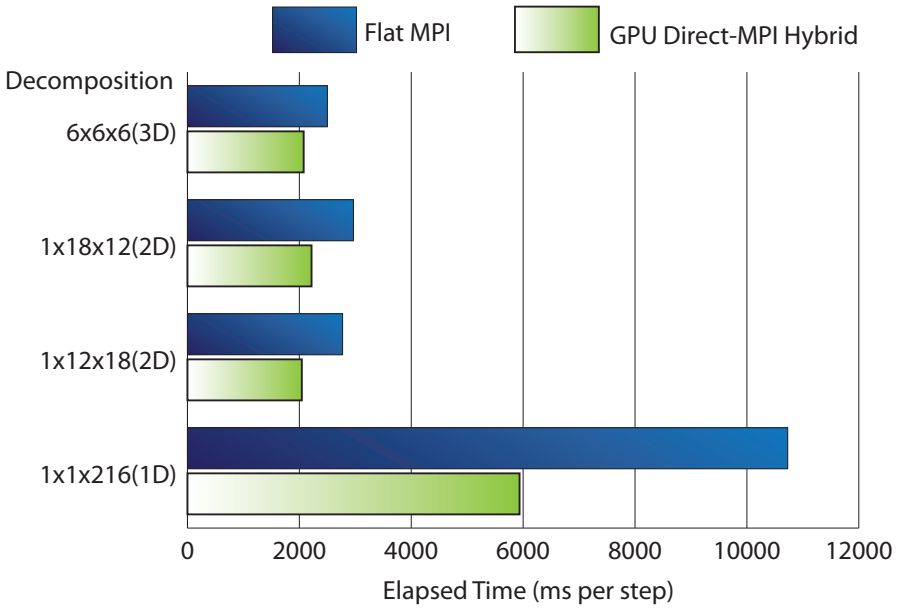

FIGURE 3.17: Comparison of different decomposition methods with respect to the elapse time on TSUBAME 2.5 (Resolution: 1296^3).

simulation on TSUBAME 2.0 of Tokyo Institute of Technology showing 2× the peak performance compared to a common flat MPI 2D decomposition approach. In the fall of 2013, TSUBAME had been upgraded to TSUBAME 2.5. All the GPUs had been replaced with more powerful Kelper generation K20X GPUs. The following Table 3.4 and Figure 3.17 show the performance tests on TSUBAME 2.5, which are similar to Table 3.2 and Figure 3.15. Since K20X contains more memory, the resolution of the computational domain of the tests are enlarged to 1296^3 (it was 864^3 in [75]).

Significant speedup of our GPU Direct-MPI hybrid communication compared to flat MPI are also shown in the experiment results of TSUBAME 2.5. However, the maximum speed is obtained by the 2D decomposition ($1 \times 12 \times 18$) where as the 3D decomposition was the fastest in our tests running on TSUBAME 2.0. We conjecture that the reason is the number of GPUs of peer-to-peer communication and the amount of data exchange. Each node of TSUBAME contains 3 GPUs. 3D decomposition only has 2 intra-node communication groups in z-direction where the $1 \times 12 \times 18$ decomposition has 6 groups. On the other hand, in the 1D decomposition ($1 \times 1 \times 216$), the GPU Direct-MPI hybrid

communication achieves 1.8× speedup which is much higher than the speedup ratio of the 1D decomposition test in [75] (1.47×). It is evident that GPU Direct-MPI hybrid parallel approach achieves high performance gain in the tests on TSUBAME 2.5 with the resolution of 1296^3. Furthermore, another 2D decomposition ($1 \times 18 \times 12$) which has less intra-node communication groups, performed slower than the 3D decomposition, which also supports our theory. 3D decomposition requires less memory for the halo. The maximum resolution of using 216 GPUs for our ideal MHD simulation is 1500^3 and achieves 2.92 TFLOPS. For comparison, the performance of the maximum resolution of using 216 GPUs for our ideal MHD simulation on TSUBAME 2.0 — 1200^3 is also tested. Performance test shows 2.1 TFLOPS on TSUBAME 2.5 where it is not a great improvement compared to TSUBAME 2.0 (2 TFLOPS). TSUBAME upgraded the GPUs but the other components are the same as TSUBAME 2.0. The computation speed and the amount of device memory of GPUs are largely improved but hardware related to data communication speed is the same. Therefore, we conclude that the similar performance is attributed to the data communication.

Figure 3.18 shows the percentage of the elapsed time between the calculation and the data communication of the break-down test of our simulation on each node. It can be realized that the data communication spend longer than the calculation even though we optimized the data communication a lot. We considering the inter-node data copy is done via MPI with H2D and D2H copy cause the overhead. Therefore, we're looking forward to use GPU Direct RDMA for fast inter-node data communication in the future when the feature is enable after the next upgrade of TSUBAME.

3.5 Summary

Performing large-scale MHD simulations on distributed multi-GPU systems provide us an opportunity to obtain the simulation results more efficiently. However, the overhead caused by the data transfer limits the simulation performance. In this section, a GPU Direct-MPI hybrid framework is proposed in order to address this problem. The following strategies are used to overcome this problem, which will fully exploit the massive computational power of GPUs :

1. We use GPU Direct 2.0 for peer-to-peer GPU data transfers to speedup the data exchange between GPUs inside each node and reduce the total number of MPI data exchanges;

2. We design CUDA kernels instead of using memory copy for copying the fragmented data.

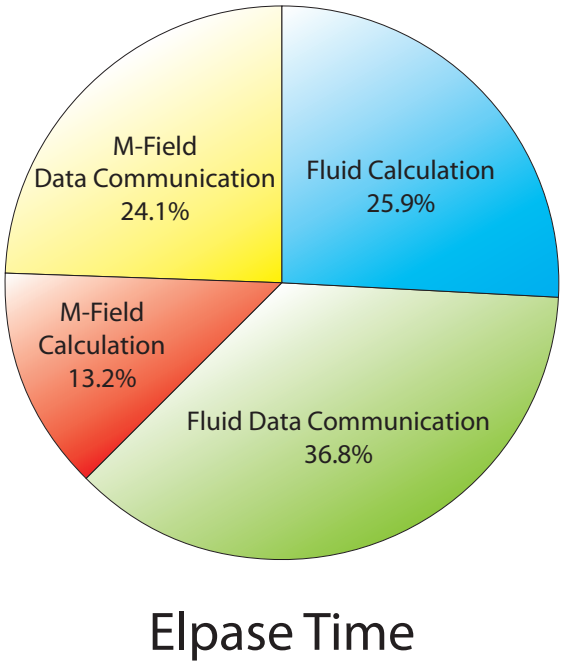

FIGURE 3.18: Percentage of the elapsed time of each process(M-Field : magnetic field).

These two strategies accelerate the data exchange along the x-direction and overcome the problem of the 3D decomposition of MHD simulations on distributed multi-GPU systems. Furthermore, since the 3D decomposition is available, it reduces the memory usage for the halo data. More data can be stored and calculated in every GPU, thus improving the overall efficiency. An Ideal MHD simulation was presented using the GPU Direct-MPI hybrid framework, which achieved 2 TFLOPS on TSUBAME 2.0 and 2.92 TFLOPS on TSUBAME 2.5 in double precision using 216 GPUs, respectively. Both the performance measurements on TSUBAME 2.0 and upgraded 2.5 show a significant enhancement with our approach compared to that of a flat MPI implementation. To achieve higher efficiency, we're looking forward to apply GPU Direct RDMA for efficient inter-node data communication.

Chapter 4

Large-Scale Global MHD Simulation

4.1 Large-scale Global MHD Simulation for Solar Wind-Earth's Magnetic Field Interaction

Studies in the space plasma environment are necessary for space exploration. The electromagnetic effect of natural phenomenon such as solar wind shock waves can damage the electronic components of an observation satellite. Although 3D global MHD models are very powerful for studying the solar wind-magnetosphere interaction, performing 3D global MHD simulations is very computationally expensive. Thus these models are often implemented on massively parallel computers [76–82]. The rapid advances in the development of graphics processing units (GPUs) provide us an alternative choice to implement global MHD models on GPUs rather than on central processing unit (CPU)-based parallel architecture.

We hereby present an efficient 3D global MHD simulator on multiple GPUs using peer-to-peer GPU communication (GPUDirect 2.0) [83]. Our method is based on the 3D global MHD model proposed by Ogino *et al.* [7, 23, 24] for simulating the interaction between the solar wind and the Earth's magnetosphere. For applications to large-scale space simulation, we extend our global MHD simulation for distributed multi-GPU systems using GPU Direct-MPI hybrid framework.

4.1.1 Global MHD Model for Simulating the Solar Wind-Magnetosphere Interaction

The global MHD model we used in our implementation was developed by Ogino *et al* [7, 23, 24] to simulate and investigate the solar wind interaction with the Earth's magnetosphere. This model has also been proven to be very useful for simulating the solar wind-planetary magnetosphere interaction, such as Jupiter's [25–28] and Saturn's magnetosphere [29–32]. The model is based on the conservation laws (density, momentum, and energy) and Maxwell equations, which form a couple of partial differential equations (PDEs) — the MHD equations. The time evolution of the solar wind-magnetosphere interaction can be simulated by solving the MHD equations. Numerical solutions of a domain (mesh) of the MHD equations along with the boundary conditions can be calculated via a numerical scheme such as the two-step Lax-Wendroff scheme [84]. In our code, the modified leapfrog scheme [24] has been implemented for performing 3D global MHD simulations on multiple GPUs. Our multi-GPU simulator is based on the following normalized resistive MHD equations 4.1, [7, 23, 24]:

$$\begin{aligned}
\frac{\partial \rho}{\partial t} &= -\nabla \cdot (\mathbf{v}\rho) + D\nabla^2\rho \\
\frac{\partial \mathbf{v}}{\partial t} &= -(\mathbf{v}\cdot\nabla)\mathbf{v} - \frac{1}{\rho}\nabla p + \frac{1}{\rho}(\mathbf{J}\times\mathbf{B}) + \mathbf{g} + \frac{\Phi}{\rho} \\
\frac{\partial p}{\partial t} &= -(\mathbf{v}\cdot\nabla)p - \gamma p\nabla\cdot\mathbf{v} + D_p\nabla^2 p \\
\frac{\partial \mathbf{B}}{\partial t} &= \nabla\times(\mathbf{v}\times\mathbf{B}) + \eta\nabla^2\mathbf{B} \\
\mathbf{J} &= \nabla\times(\mathbf{B}-\mathbf{B}_d)
\end{aligned} \tag{4.1}$$

where ρ is the plasma density, $\mathbf{v}$ the velocity, $\mathbf{B}$ the magnetic field vector, and p the plasma pressure. The fifth equation of Equation 4.1 is adopted to removed the numerical error of a finite difference method. $\mathbf{B}_d$ 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 $0.0001 \leq \eta \leq 0.002$ for the magnetospheric configuration, $\mathbf{g} = g_0/\xi^3$ ($\xi = \sqrt{x^2+y^2+z^2}$, g_0 is the standard gravity (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 oscillation of the short wavelength resulting from an initial value or a rapid magnetic field change,

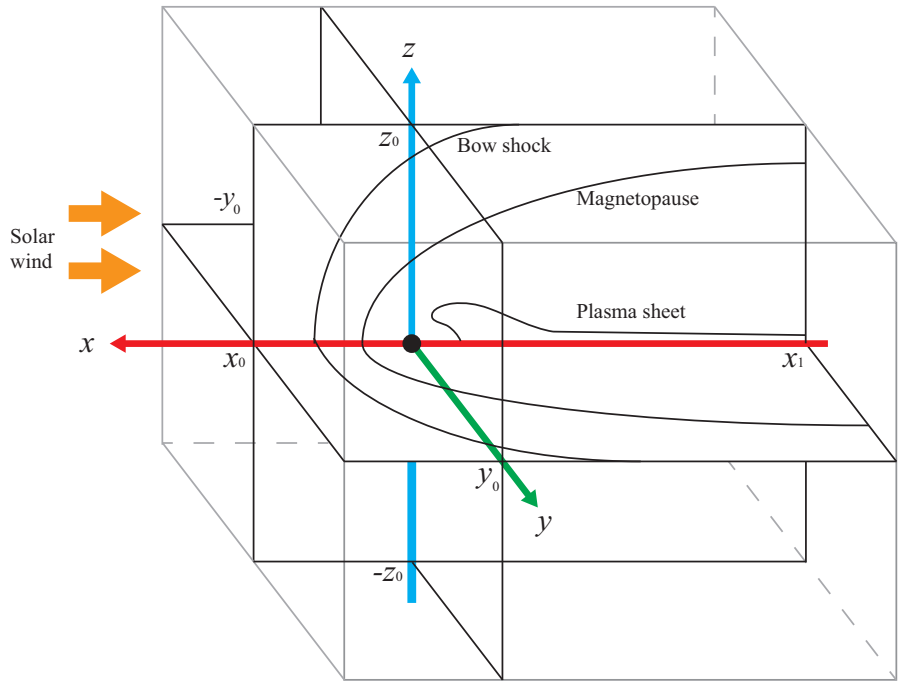

FIGURE 4.1: Simulation domain of solar wind interacting with the Earth's magnetosphere. The Sun is located in the positive side along the x-axis, and the north magnetic pole is assumed along the z-axis positive side.

$D = D_p = \mu/\rho_{sw} = 0.001$, where ρ_{sw} is the solar wind density. $R_e = 6.37 \times 10^3$ km is the radius of the Earth.

The calculation domain of our global MHD simulation of solar wind-magnetosphere interaction is shown in Figure 4.1. The whole computational domain of the simulation is $(x, y, z) = (-60R_e, -30R_e, -30R_e) \sim (30R_e, 30R_e, 30R_e)$. The Earth is placed at the origin of the coordinate system. The initial condition is set to a constant source of density, velocity, and temperature from the upper stream ($x = x_0$) of the simulation domain. The magnetic field of the Earth is defined as $\mathbf{B}_d = (\frac{-3xz}{\xi^5}, \frac{-3yz}{\xi^5}, \frac{x^2+y^2-z^2}{\xi^5})$. The parameter of the solar wind is set as follows: density $\rho_{sw} = 5 \times 10^{-4}$ (equivalent to $5/\text{cm}^{-3}$), $\mathbf{v}_{sw} = (v_{sw}, 0, 0), v_{sw} = 0.0627$ to 0.117(400 to 750 km/s), $p_{sw} = 3.56 \times 10^{-8}(T_{sw} = 2 \times 10^5$ K) at the upper stream ($x = x_0$). The interplanetary magnetic field (IMF) is set to $\mathbf{B}_{IMF} = -1.5 \times 10^{-4}$ (-5 nT) for in the following tests.

The ionospheric boundary condition is imposed surrounding the Earth [23]. Solar wind comes as the constant source along the x-axis from the upstream boundary at $x = x_0 = 30R_e$ to the free/open(Neumann) boundary at $x = x_1 = -60R_e$. At $y = \pm y_0$ and $z = \pm z_0$ the free/open boundaries are set to the direction with 45 degrees to the x-axis.

The implementation of the global MHD simulation is straightforward. GPU Direct-MPI hybrid framework will handle the data decomposition and the data communication

between GPUs of the distributed multi-GPU system. Other issues such as file I/O can be the same, follow, or even reuse from the ideal MHD simulation. The only differences are the numerical scheme, the parameter setting of the model and the boundary conditions.

4.1.2 GPU Implementation of the Modified Leapfrog Scheme for Global MHD Simulation

The modified leapfrog method [24] is one of the finite difference method. Finite difference methods have ease of computational complexity but at the cost of stability. In preliminary analysis to our work [57, 75, 83], we found that the computational complexity of the modified leapfrog method is about a half of the relaxing TVD method. On the other hand, the relaxing TVD method requires 2 halos in each direction because of the magnetic field is set on the cell face, where ther modified leapfrog method require 1 halo only since all the quantities is cell-centered. In addition, The dimensional splitting of the relaxing TVD requires more data communication processes in each calculation step. The relaxing TVD requires data communication in the calculation of each direction (each L_i, described in Section 2.2). Therefore it requires 6 data communications in one calculation step, where the modified leapfrog method only requires one data communication per step. Since efficiency are memory usage are demanding to reach our goal of large-scale simulation as well as the in-situ visualization, we chose the modified leapfrog method instead of the relaxing TVD method for the global MHD simulation. In fact, our global MHD simulations using the modified leapfrog achieve 4.38 TFLOPS in double precision, where our ideal MHD simulations using the relaxing TVD method only achieve 2.92 TFLOPS. As we explained in Section 4.3, In-situ visualization of our global MHD simulations achieve real-time frame rate. However, it is not efficient enough to provide real-time visualization with user interaction if we used the relaxing TVD method.

The modified leapfrog method is a combination of the two-step Lax-Wendroff scheme [84] and the leapfrog scheme [84]. 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 [24] suggest that $l = 8$ is the optimal number for the modified leapfrog scheme to provide stable results. For the detail of the modified leapfrog scheme, please refer to Appendix B.

The modified leapfrog scheme is used to solve the MHD equations and Maxwell's equations. In the following description, in order to avoid confusion between the index and the direction, data of a grid-point will be shown as $v_x(i,j,k,t)$, where (i,j,k) is a 3D index of the mesh (they actually have the same direction as (x,y,z)). Then, we can express the discretized equation of v_x. We have the 1st step of the numerical scheme of v_x of Equation 4.1 as follows:

$$\begin{aligned} v_x(i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2},t+\frac{1}{2}) =& v_x(I) - \frac{\Delta t}{4\Delta x} v_x(I) D_x[v_x(i,j,k,t)] \\ & - \frac{\Delta t}{4\Delta y} v_y(I) D_y[v_x(i,j,k,t)] \\ & - \frac{\Delta t}{4\Delta z} v_z(I) D_z[v_x(i,j,k,t)] \\ & - \frac{\Delta t}{4\Delta x} D_x[p(i,j,k,t)] \frac{1}{\rho(I)} \\ & + \frac{\Delta t}{\rho(I)} [J_y(i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2},t) B_z(I) \\ & - J_z(i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2},t) B_y(I)] + \alpha \end{aligned} \tag{4.2}$$

where we let α represent the gravity term and viscous term for brevity. Index $I = (i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2}, t)$ for the two-step Lax-Wendroff scheme and $I = (i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2}, t-\frac{1}{2})$ for the leapfrog scheme. $D_x(\phi)$, $D_y(\phi)$, and $D_z(\phi)$ are the functions for calculating the differences in x, y and z directions for any physics quantity $\phi(i,j,k,t)$:

$$\begin{aligned} D_x(\phi(i,j,k,t)) =& \phi(i+1,j+1,k+1,t) + \phi(i+1,j+1,k,t) \\ & + \phi(i+1,j,k+1,t) + \phi(i+1,j,k,t) \\ & - \phi(i,j+1,k+1,t) - \phi(i,j+1,k,t) \\ & - \phi(i,j,k+1,t) - \phi(i,j,k,t) \end{aligned}$$

$$\begin{aligned} D_y(\phi(i,j,k,t)) =& \phi(i+1,j+1,k+1,t) + \phi(i+1,j+1,k,t) \\ & + \phi(i,j+1,k+1,t) + \phi(i,j+1,k,t) \\ & - \phi(i+1,j,k+1,t) - \phi(i+1,j,k,t) \\ & - \phi(i,j,k+1,t) - \phi(i,j,k,t) \end{aligned} \tag{4.3}$$

$$\begin{aligned} D_z(\phi(i,j,k,t)) =& \phi(i+1,j+1,k+1,t) + \phi(i+1,j,k+1,t) \\ & + \phi(i,j+1,k+1,t) + \phi(i,j,k+1,t) \\ & - \phi(i+1,j+1,k,t) - \phi(i+1,j,k,t) \\ & - \phi(i,j+1,k,t) - \phi(i,j,k,t) \end{aligned}$$

From the above equations one can easily see that there are many inputs for the calculation kernels. This situation is problem for a CPU program, however, as we mentioned above, such huge number of inputs will cause the CUDA kernels to underperform. It is an important issue that needs to be addressed when implementing a GPU program. Actually, this issue can be solved straightforward by splitting Equation (4.2) into many short equations (steps) as follows:

$$\begin{aligned} v_x(i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2},t+\frac{1}{2}) =& v_x(I) - \frac{\Delta t}{4\Delta x} v_x(I) D_x[v_x(i,j,k,t)] \\ & - \frac{\Delta t}{4\Delta y} v_y(I) D_y[v_x(i,j,k,t)] \\ & - \frac{\Delta t}{4\Delta z} v_z(I) D_z[v_x(i,j,k,t)] \end{aligned} \tag{4.4}$$

$$\begin{aligned} v_x(i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2},t+\frac{1}{2}) =& v_x(i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2},t+\frac{1}{2}) \\ & - \frac{\Delta t}{4\Delta x} D_x[p(i,j,k,t)] \frac{1}{\rho(I)} \end{aligned} \tag{4.5}$$

$$\begin{aligned} v_x(i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2},t+\frac{1}{2}) =& v_x(i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2},t+\frac{1}{2}) \\ & + \frac{\Delta t}{\rho(I)} [J_y(i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2},t) B_z(I) \\ & - J_z(i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2},t) B_y(I)] + \alpha \end{aligned} \tag{4.6}$$

Equation (4.2) is now split into three steps (Equations (4.4), (4.5), and (4.6)). To improve the efficiency, we have tried our best to minimise the number of the input parameters of each kernel (function) call. The same method is applied to other equations in every step in the modified leapfrog scheme. Beside the initialization, the overall program flow is shown as follows, noted that the physics quantity is $\phi = \{\rho, \mathbf{v}, p, \mathbf{B}\}$:

For large-scale global MHD simulation using distributed multi-GPU system, our GPU Direct-MPI hybrid framework which is introduced in the previous section is used. Data communication between the partitions stored on the dedicated GRAM of different GPU is handled using our framework to achieve high efficiency. On the other hand, the computational complexity and required halo size of the modified leapfrog scheme is lower then the relaxing TVD scheme. Thus, the elapse time per step is faster.

4.2 Experimental Results

Large-scale global MHD simulations were run using the TSUBAME 2.5 supercomputer of Tokyo Institute of Technology. In this section, experiments of solar wind interacting with a magnetic dipole field of a planet is presented. The first test is the solar wind interacting

Algorithm 3 GPU implementation of the modified leapforg scheme

1: Exchange the data at the HALO with the adjacent partition domain.
2: Assign boundary conditions and calculate the boundary data;
3: Calculate the current, $\mathbf{J} \leftarrow \mathbf{B}(t)$;
4: `If` $l = 1$, interpolate $\phi(i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2}, t)$ and proceed with the two-step Lax-Wendroff scheme.
`Else` use $\phi(i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2}, t-\frac{1}{2})$ to proceed with the leapfrog scheme;
5: The 1st step of the numerical scheme,
`If` $l = 1$, two-step Lax-Wendroff scheme
$\phi(i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2}, t+\frac{1}{2}) \leftarrow [\mathbf{J}, \phi(i,j,k,t), \phi(i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2}, t)$
`Else`, the leapfrog scheme
$\phi(i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2}, t+\frac{1}{2}) \leftarrow [\mathbf{J}, \phi(i,j,k,t), \phi(i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2}, t-\frac{1}{2})]$;
6: Calculate the current, $\mathbf{J} \leftarrow \mathbf{B}(t+\frac{1}{2})$;
7: The 2nd step of the numerical scheme, $\phi(i,k,j,t+1) \leftarrow [\mathbf{J}, \phi(i,j,k,t), \phi(i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2}, t+\frac{1}{2})]$;
8: `If` $t < t_{finish}, t = t + \Delta t, l = (l+1)$ mod 8, repeat from Step 1.
`Else` finish the simulation and output the results.

with Earth's magnetosphere. The magnetosphere is approximated by a vertical dipole — magnetic poles along to the z-axis. Phenomenon such as the bow shock, polar cusp and magnetopause were successfully reproduced in our simulation. The magnetic poles of the second test is a dipole field inclined to the z-axis. These are preliminary studies of simulating solar wind interacting with the Earth's magnetic field while geomagnetic reversal happens. Difference of the polar cusp, as well as the asymmetry plasma flow at the tail side can be found.

4.2.1 Solar Wind-Earth Magnetosphere Interaction

3D visualization (isosurface) of our simulation results of the solar wind interacting with Earth's magnetosphere are shown in Figures 4.3 to 4.6 (Total pressure : p). Figures 4.7 to 4.9 show the $x-z$ plane at $y = 0$ of the corresponding result of the total pressure as well as the magnetic field. The evolutions of the solar wind pressing the front side of the Earth's magnetic field as well as stretching the tail side are shown. The structure of the Earth's magnetic field including the bow shock, polar cusp, magnetosheath, magnetopause and magnetotail are reproduced in our simulations. Figure 4.2 shows that our simulation results are in good agreement to [7].

4.2.2 Application to Geomagnetic Reversal

Most magnetic fields of the planets in the Solar System can be approximated by a dipole field, but with an inclination (Dipole tilt). As a related matter, the theory of the geomagnetic reversal has been proposed. In this section, we present our simulation with

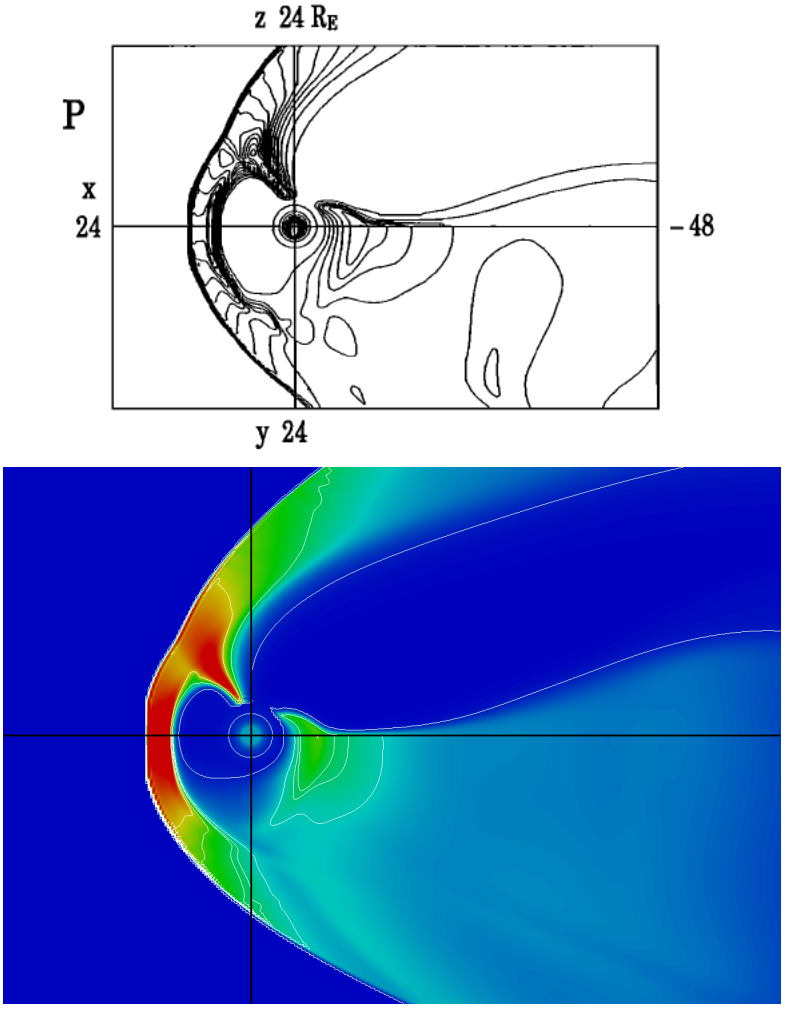

FIGURE 4.2: Comparison between our simulation results and the results presented in [7]. The upper half of the figures is the $x-z$ plane, where the lower half is the $x-y$ plane. Note that our simulation domain $((x,y,z) = (-60R_e,-30R_e,-30R_e) \sim (30R_e,30R_e,30R_e))$ is larger than the simulation domain of [7] $((x,y,z) = (-48R_e,0R_e,0R_e) \sim (24R_e,24R_e,24R_e))$

a inclined dipole field. Considering that in real practice, the magnetic field of a planet may be input from observation data rather generate with a function, we do not modified the function $\mathbf{B}_d$ but use a unit quaternion [85] to rotate the dipole field. In 3-dimensional space, according to Euler's rotation theorem, any sequence of rotations of a rigid body or coordinate system about a fixed point is equivalent to a single rotation by a given angle ϕ about a fixed axis (called *Euler axis*) that runs through the fixed point. The Euler axis is typically represented by a unit vector $\overrightarrow{u}$. A Euclidean vector (a_x, a_y, a_z) can be rewritten as $a_x\mathbf{i} + a_y\mathbf{j} + a_z\mathbf{k}$, where $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{k}$ are unit vectors representing the three Cartesian axes. A rotation through an angle of ϕ around the axis defined by a unit vector

$$\overrightarrow{u} = u_x, u_y, u_z = u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k}$$

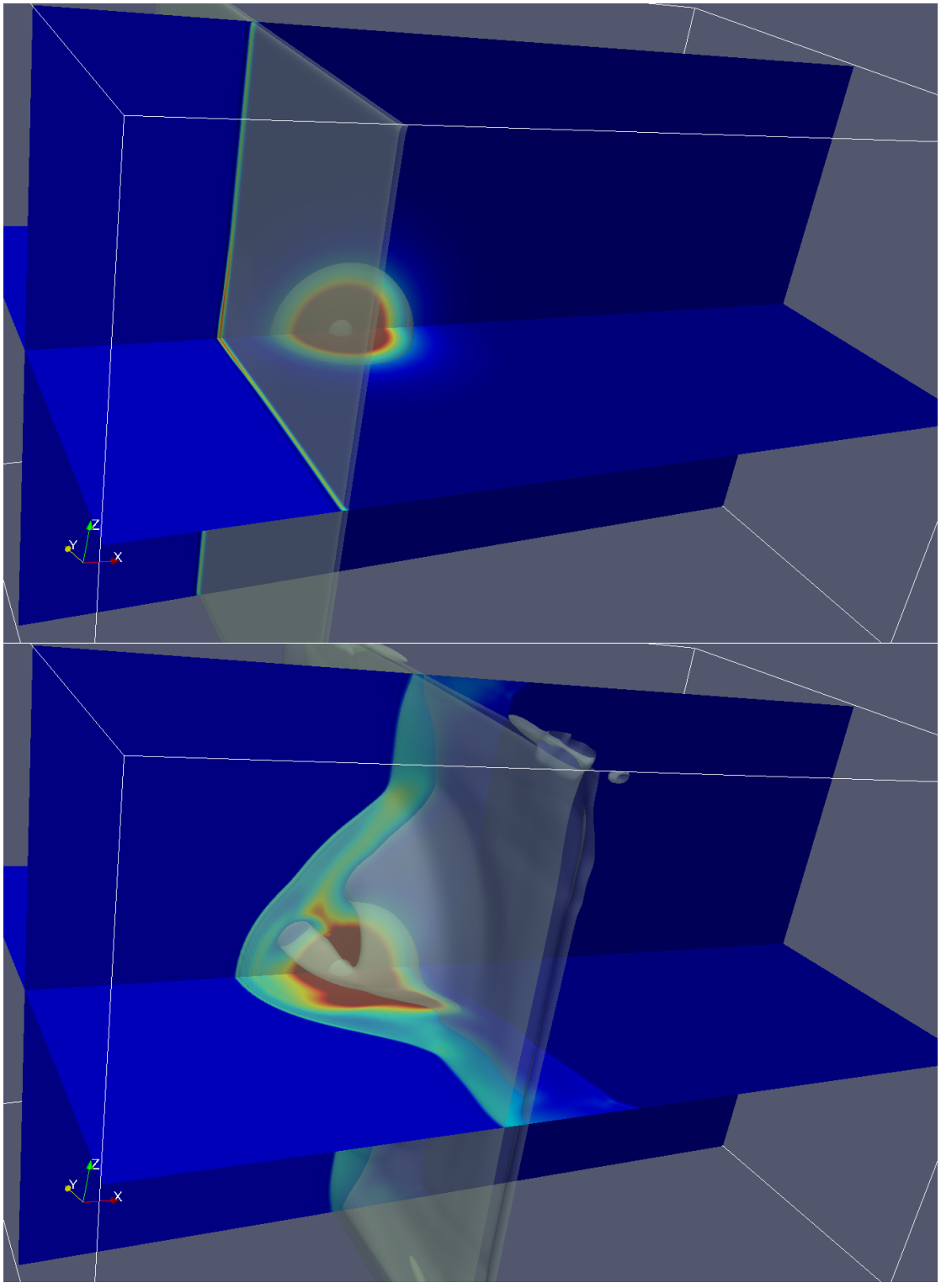

FIGURE 4.3: Results of the density (isosurface) evolution of the Solar Wind-Earth interaction (1/2).

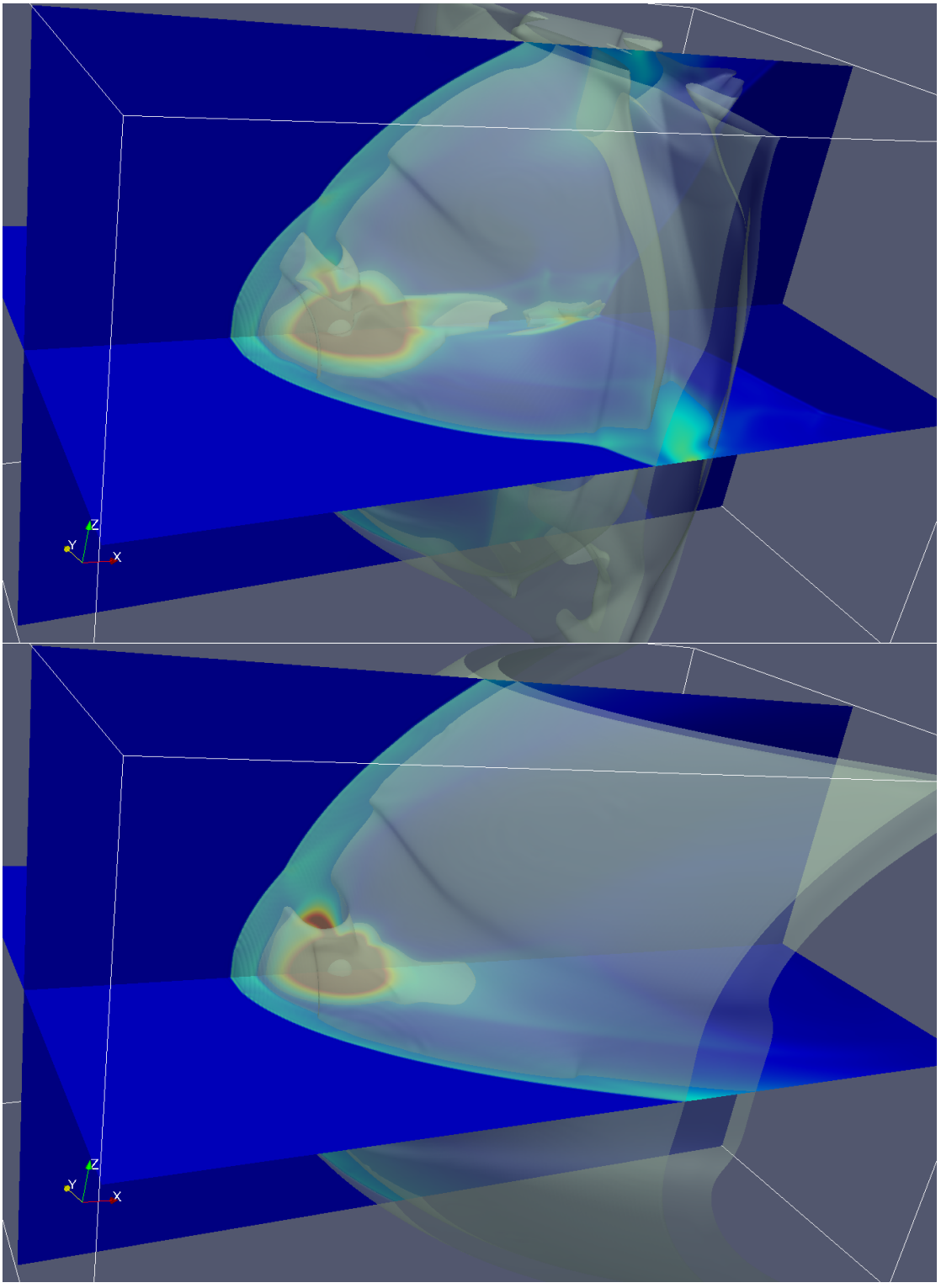

FIGURE 4.4: Results of the density (isosurface) evolution of the Solar Wind-Earth interaction (2/2).

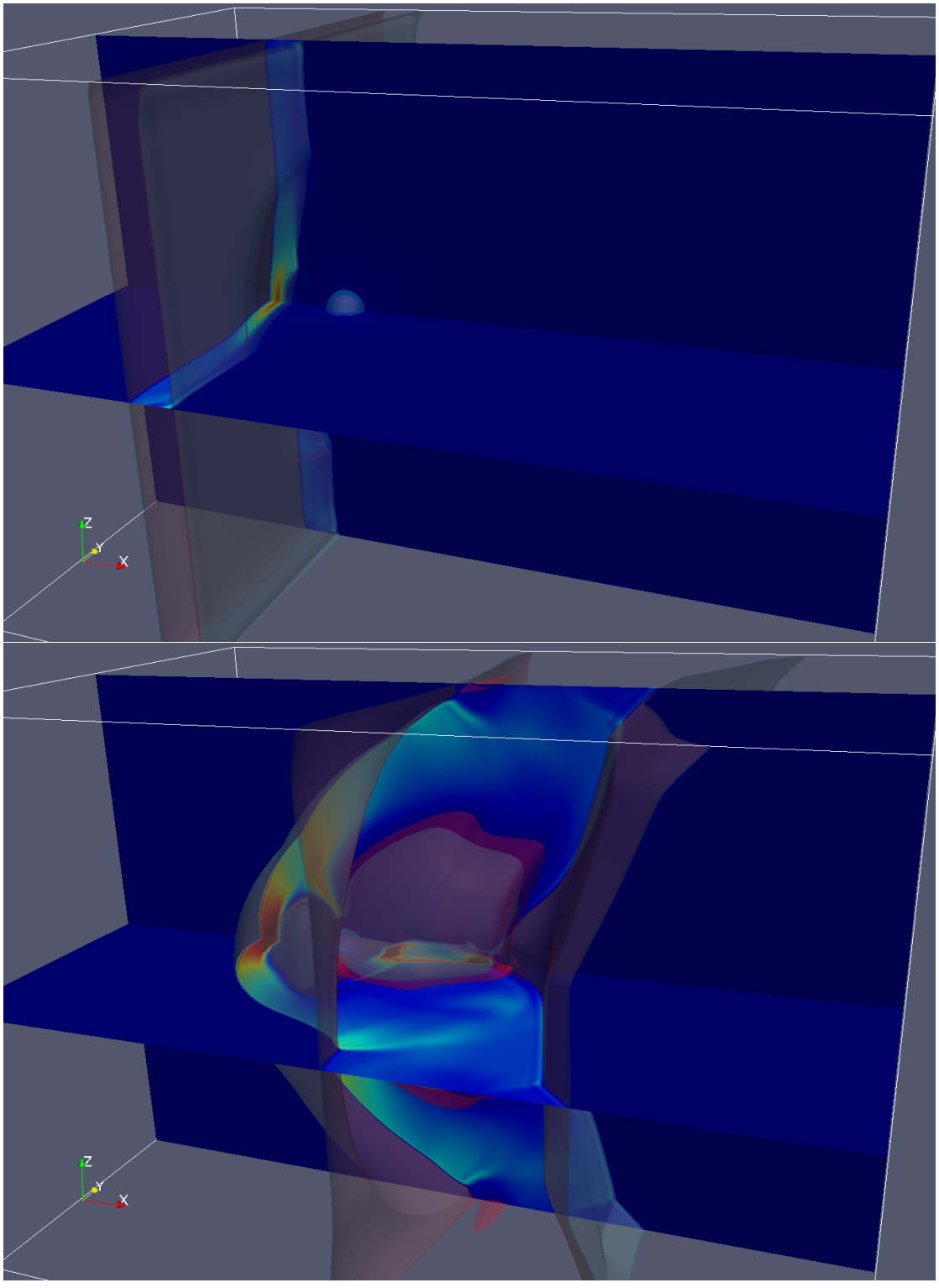

FIGURE 4.5: Results of the pressure (isosurface) evolution of the Solar Wind-Earth interaction (1/2).

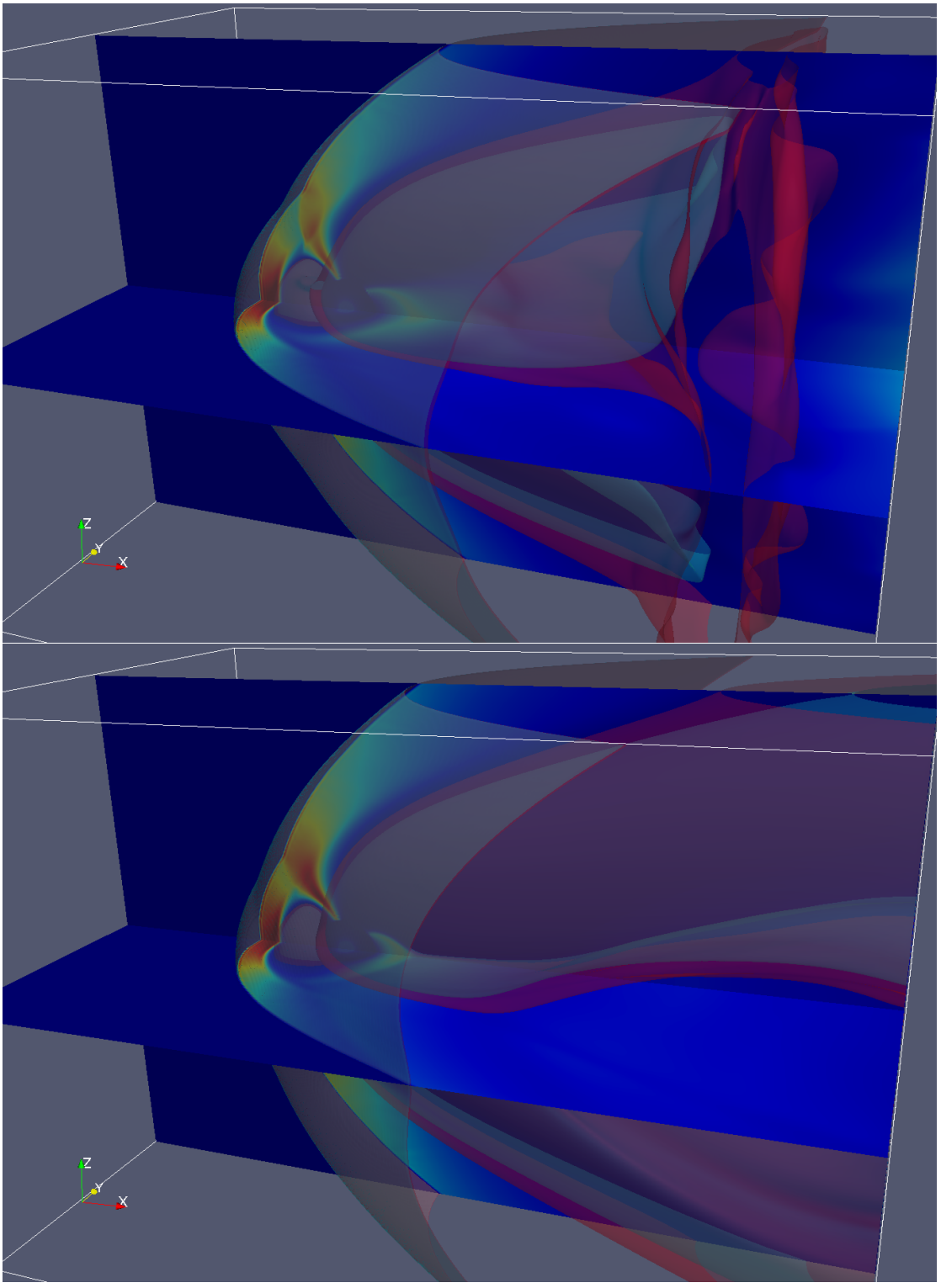

FIGURE 4.6: Results of the pressure (isosurface) evolution of the Solar Wind-Earth interaction (2/2).

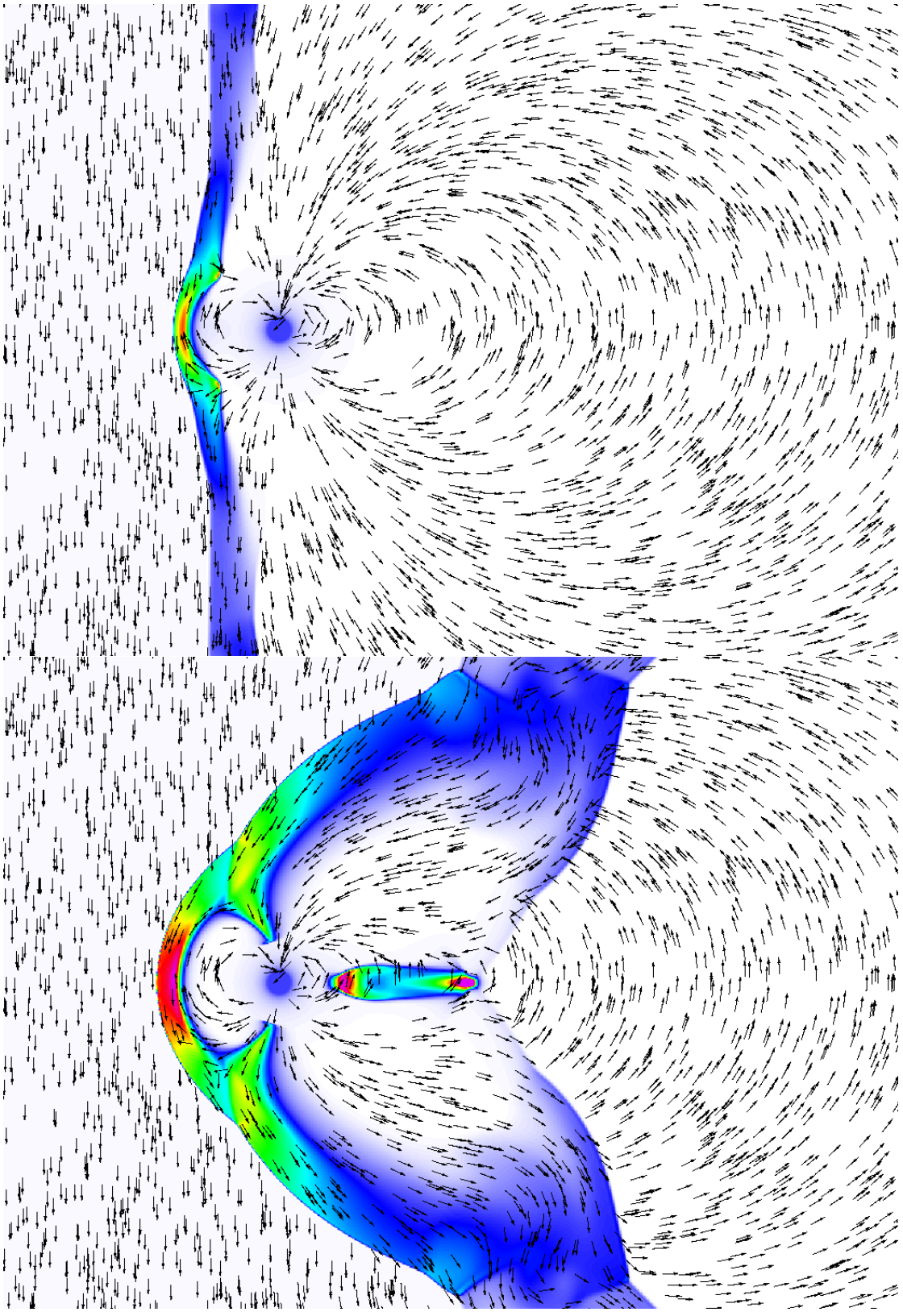

FIGURE 4.7: Results of the pressure evolution of the Solar Wind-Earth interaction (1/3).

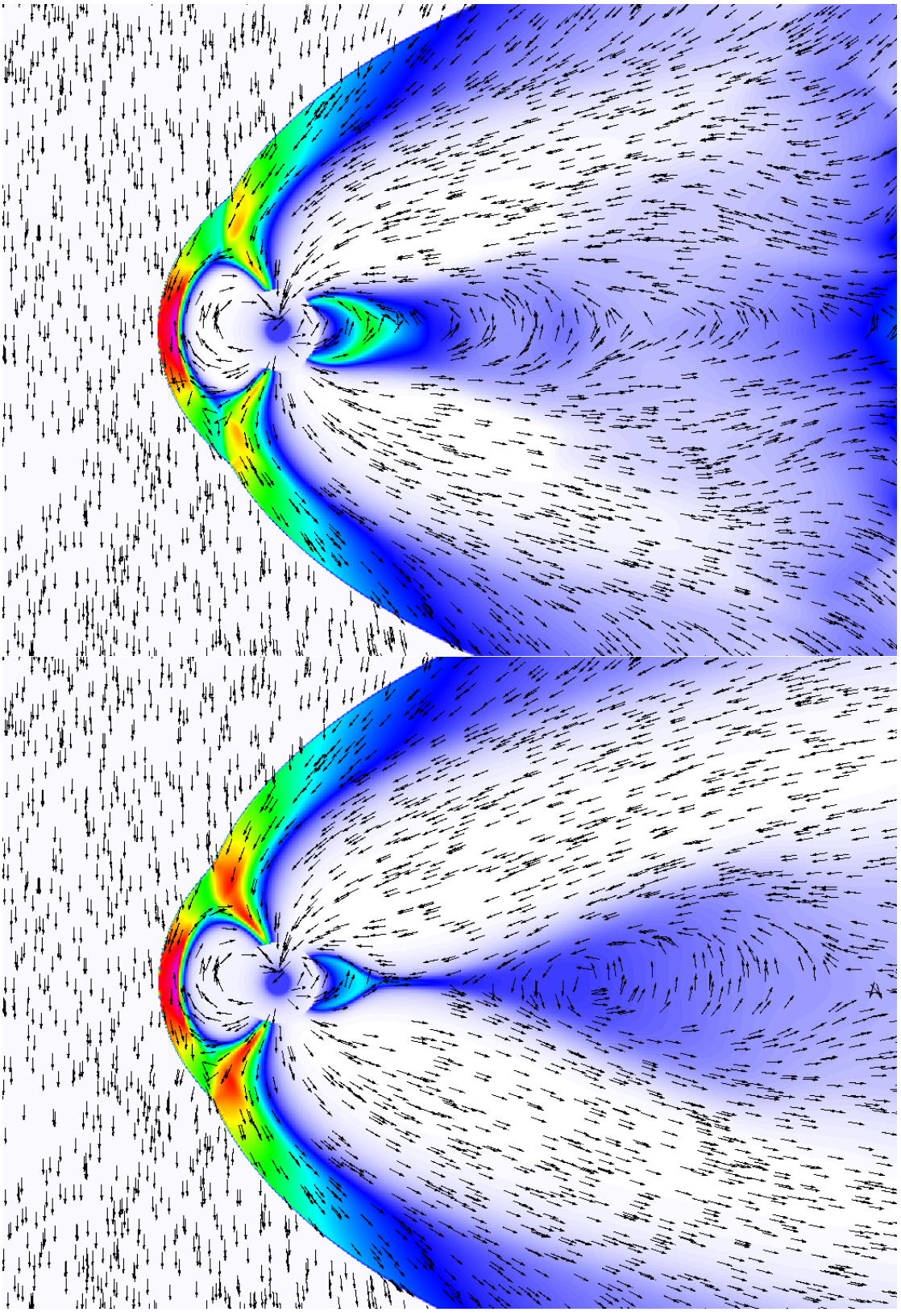

FIGURE 4.8: Results of the pressure evolution of the Solar Wind-Earth interaction (2/3).

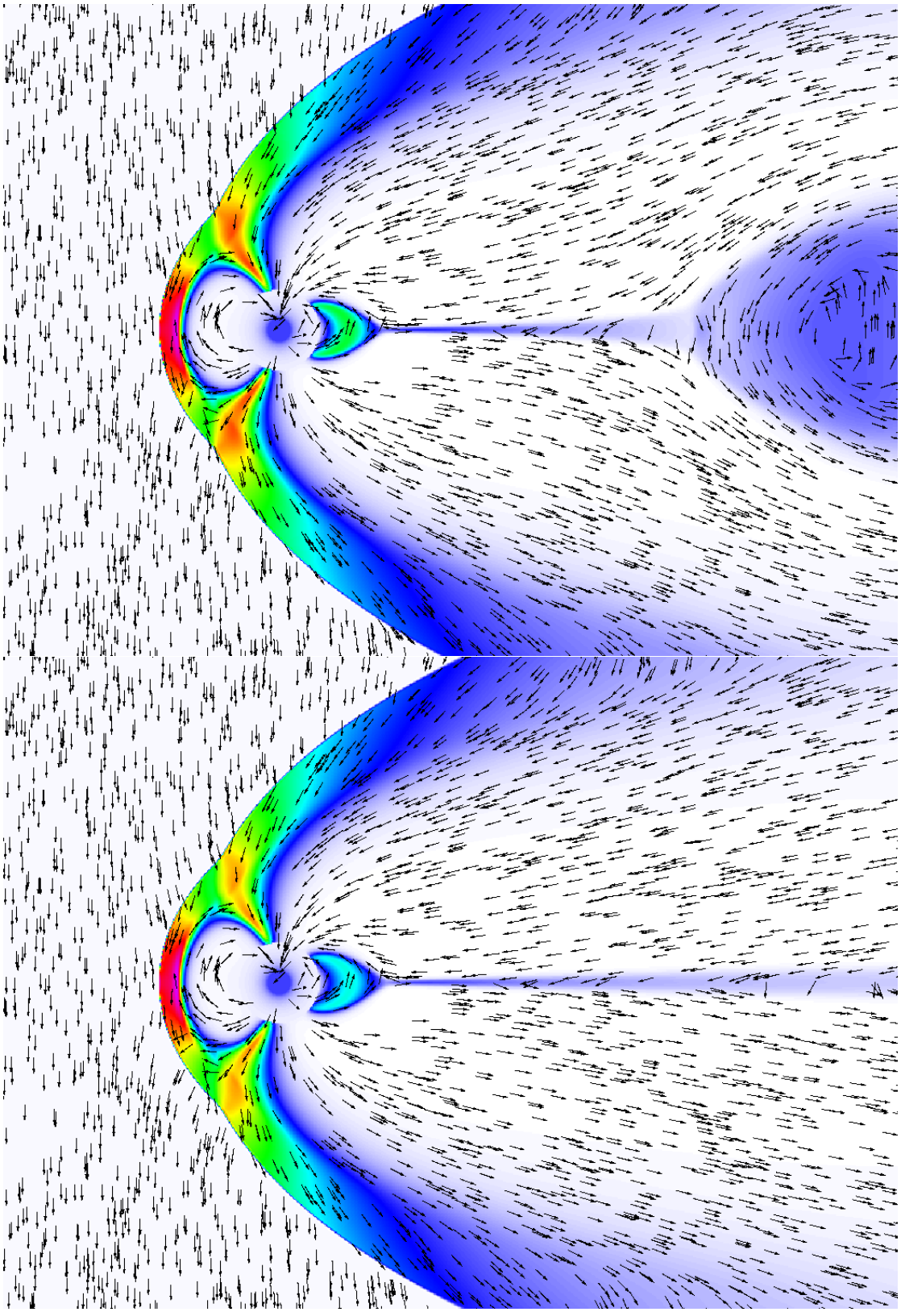

FIGURE 4.9: Results of the pressure evolution of the Solar Wind-Earth interaction (3/3).

Quaternions give a simple way to encode this four numbers. A quaternion $\mathbf{q} = (w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k})$ has an (x, y, z) component to represent the Euler axis ($\overrightarrow{u}$). The angle ϕ is presented by the w component. Then, the above unit vector can be represented by a quaternion. This can be done using an extension of Euler's formula:

$$\mathbf{q} = \exp[\frac{\phi}{2}(u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})] = \cos\frac{\phi}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\phi}{2}$$

A rotation matrix with a given axis $\overrightarrow{u}$ (The Euler axis, where $u_x^2 + u_y^2 + u_z^2 = 1$) by an angle ϕ is

$$\begin{bmatrix} \cos\phi + u_x^2(1-\cos\phi) & u_xu_y(1-\cos\phi) - u_z\sin\phi & u_xu_z(1-\cos\phi) + u_y\sin\phi \\ u_yu_x(1-\cos\phi) + u_z\sin\phi & \cos\phi + u_y^2(1-\cos\phi) & u_yu_z(1-\cos\phi) - u_x\sin\phi \\ u_zu_x(1-\cos\phi) - u_y\sin\phi & u_zu_y(1-\cos\phi) + u_x\sin\phi & \cos\phi + u_z^2(1-\cos\phi) \end{bmatrix}$$

By using a unit quaternion, the rotation matrix can be expressed as

$$\begin{bmatrix} 1-2y^2-2z^2 & 2(xy-wz) & 2(xz+wy) \\ 2(xy+wz) & 1-2x^2-2z^2 & 2(yz-wx) \\ 2(xz-wy) & 2(yz+wx) & 1-2x^2-2y^2 \end{bmatrix}$$

In the following tests, the rotation matrices had been calculated with a unit quaternion of the inclination angle and then applied to $\mathbf{B}_d$ using matrix multiplication to obtain the accurate rotated dipole field. Simulation results of the solar wind interacting with the Earth's magnetic field with inclination are presented in Figures 4.11 to 4.16.

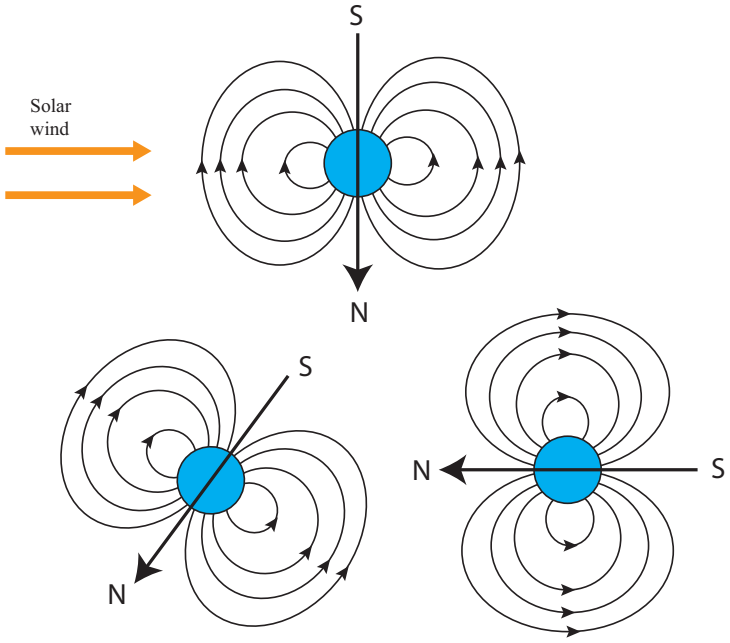

FIGURE 4.10: Dipole magnetic fields of inclined magnetic pole

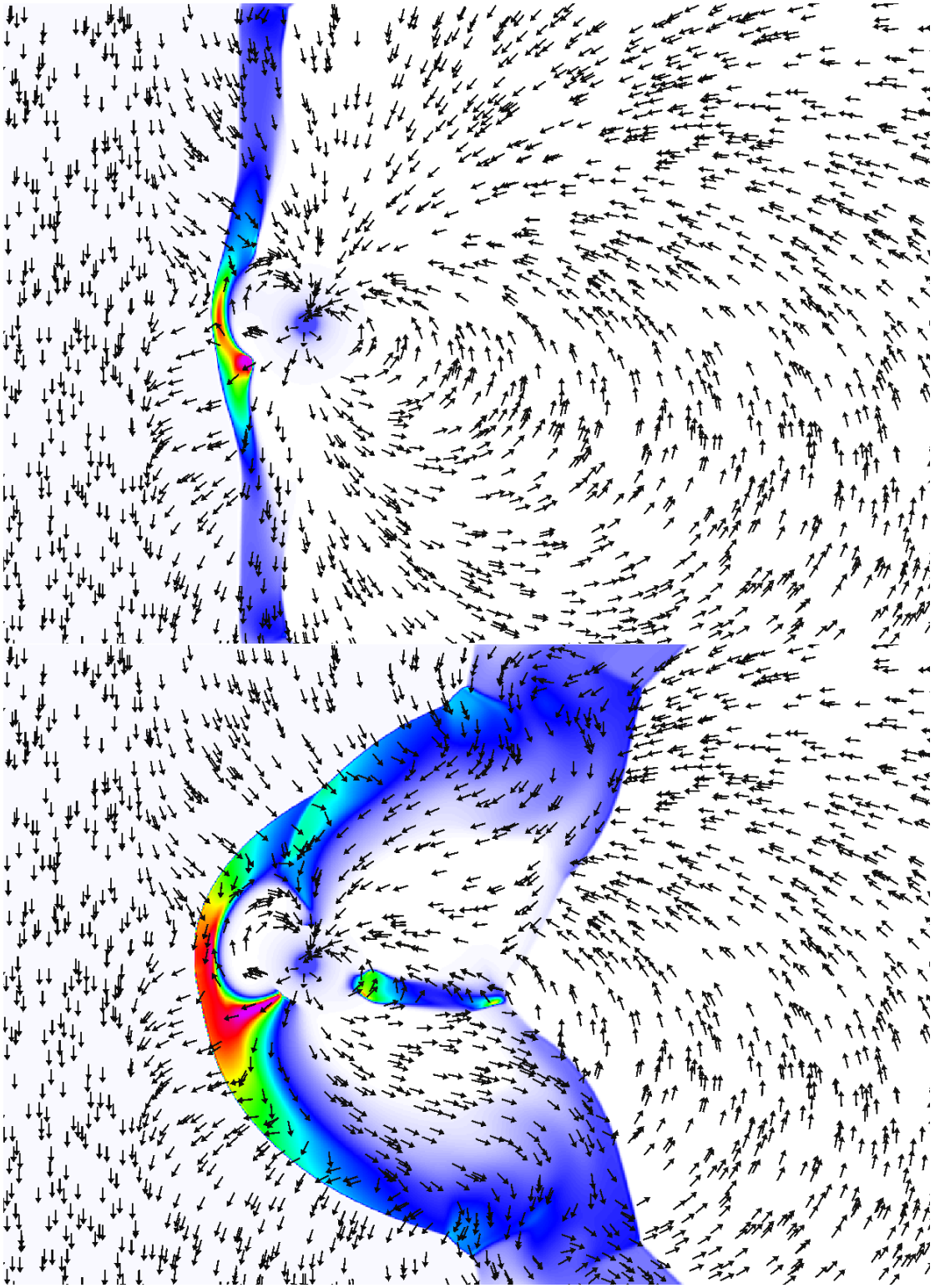

FIGURE 4.11: Results of the pressure evolution of the solar wind interaction with inclined dipole field (1/3).

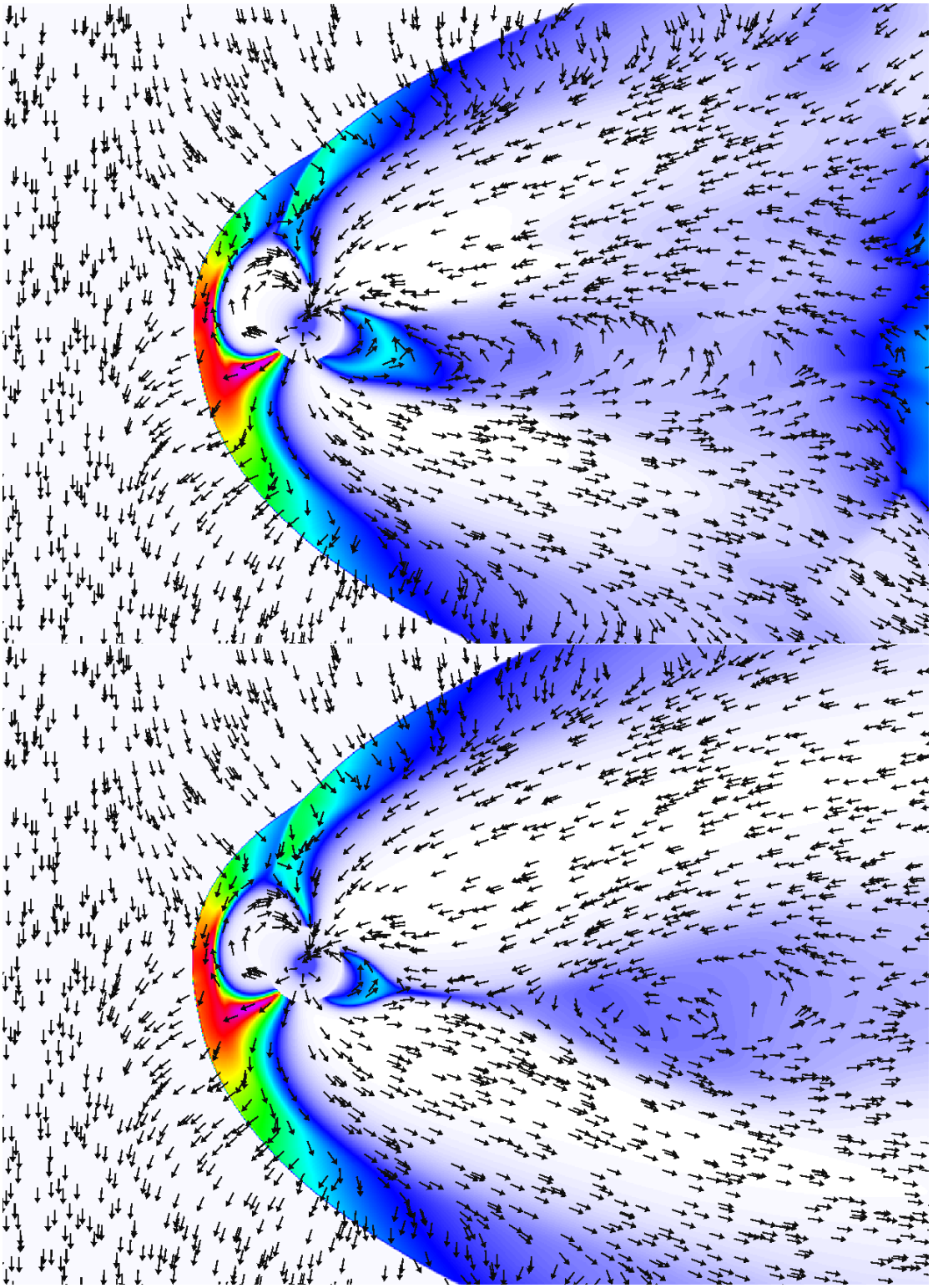

FIGURE 4.12: Results of the pressure evolution of the solar wind interaction with inclined dipole field (2/3).

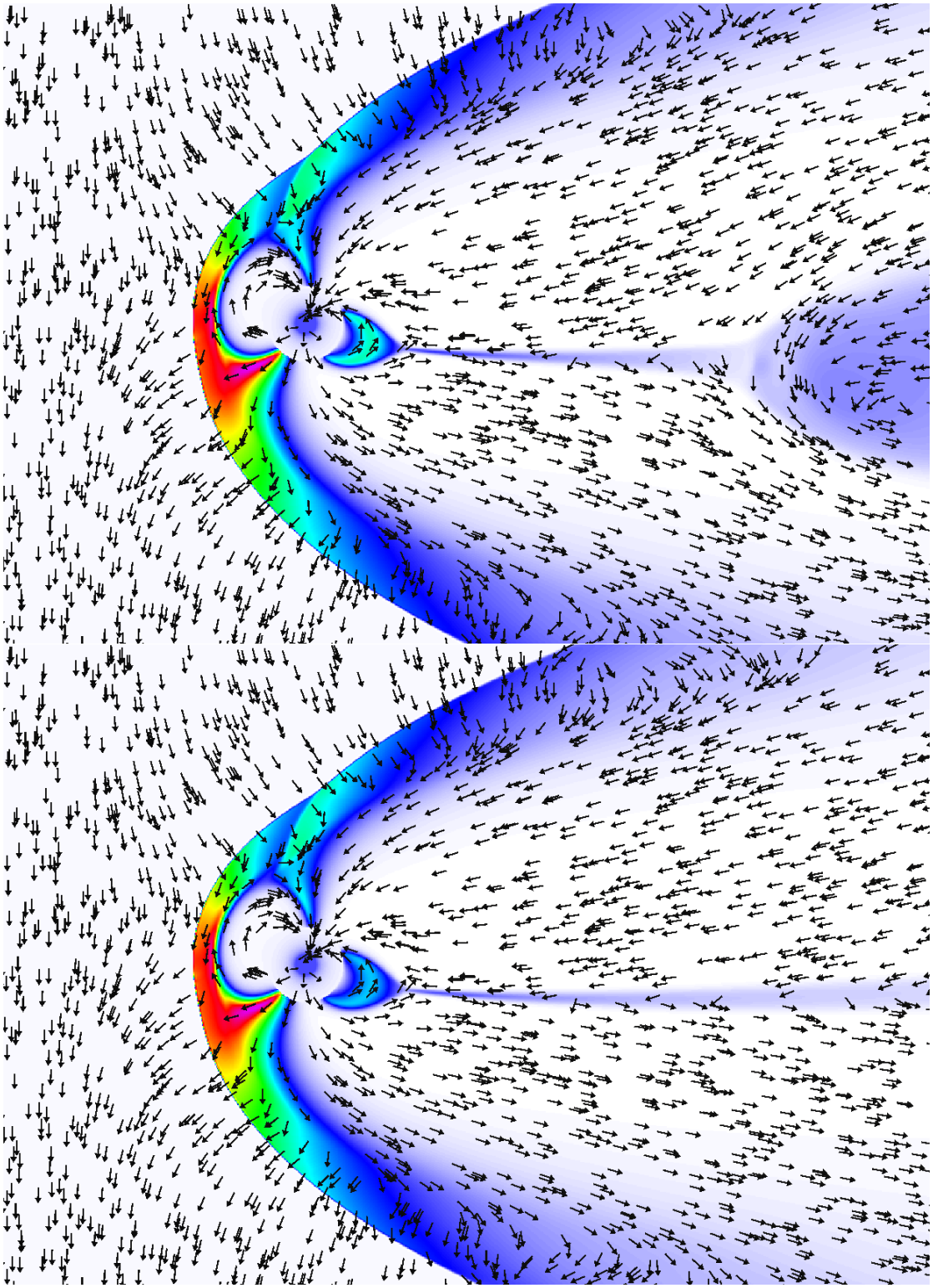

FIGURE 4.13: Results of the pressure evolution of the solar wind interaction with inclined dipole field (3/3).

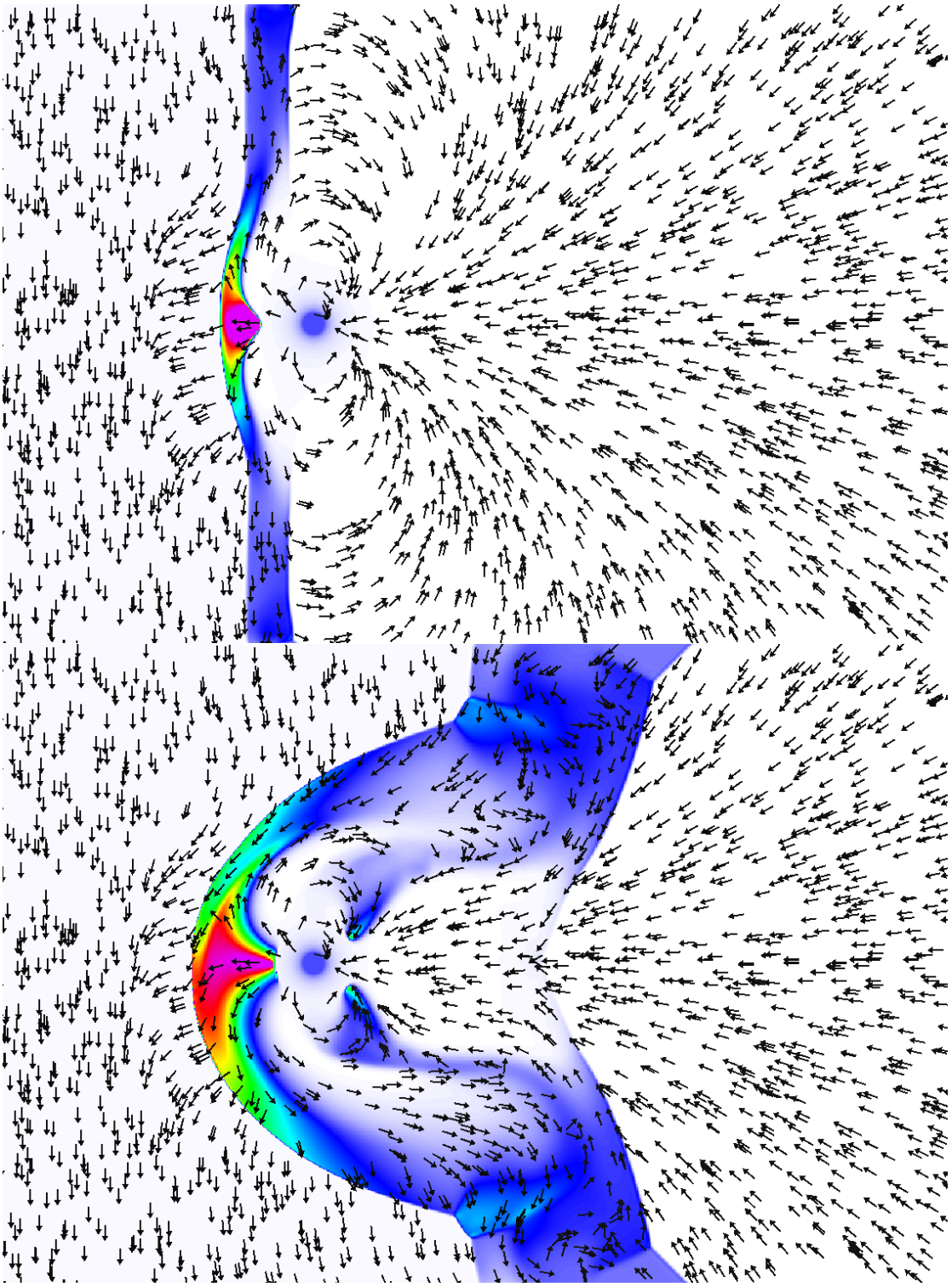

FIGURE 4.14: Results of the pressure evolution of the solar wind interaction with horizontal dipole field (1/3).

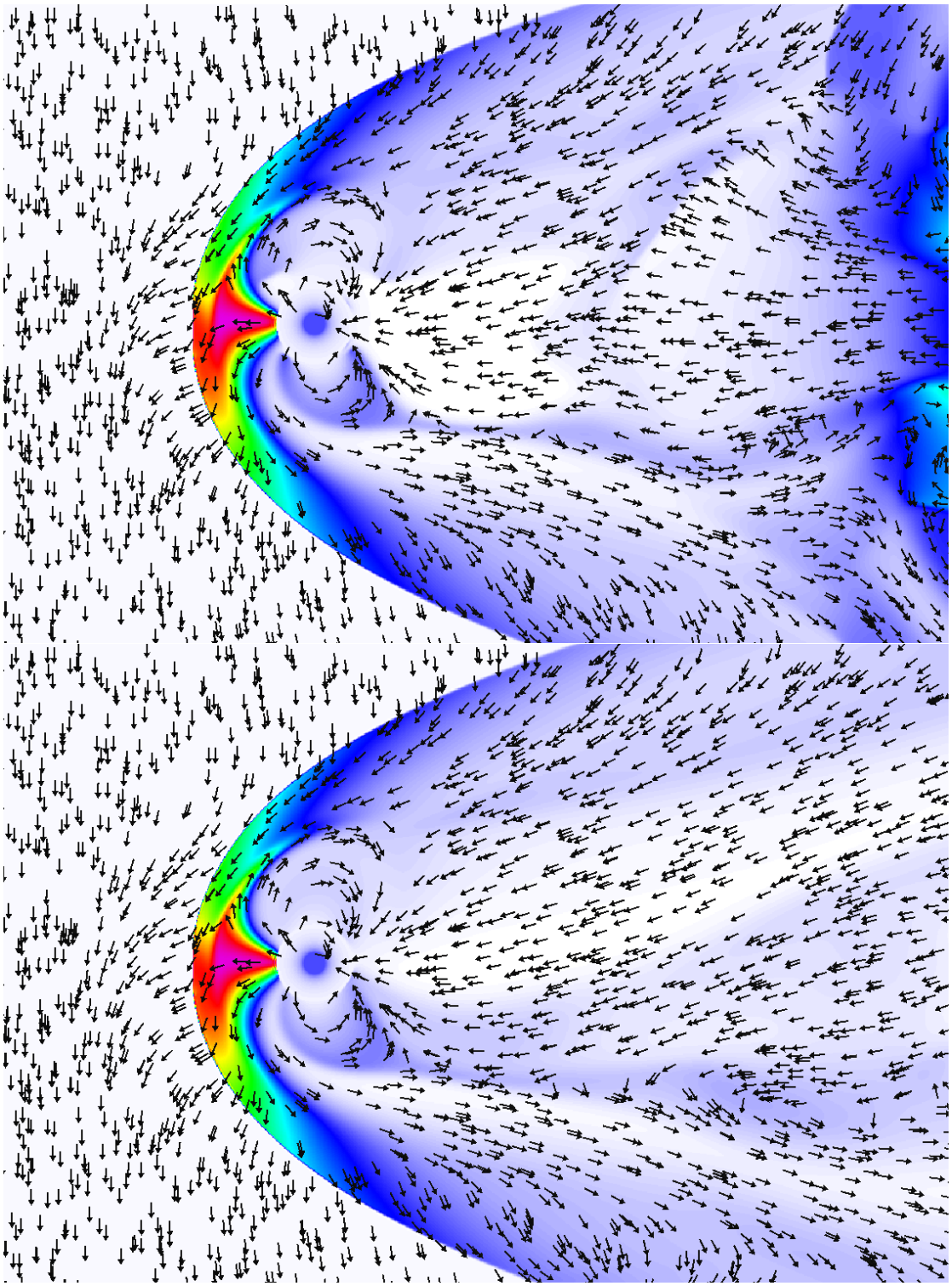

FIGURE 4.15: Results of the pressure evolution of the solar wind interaction with horizontal dipole field (2/3).

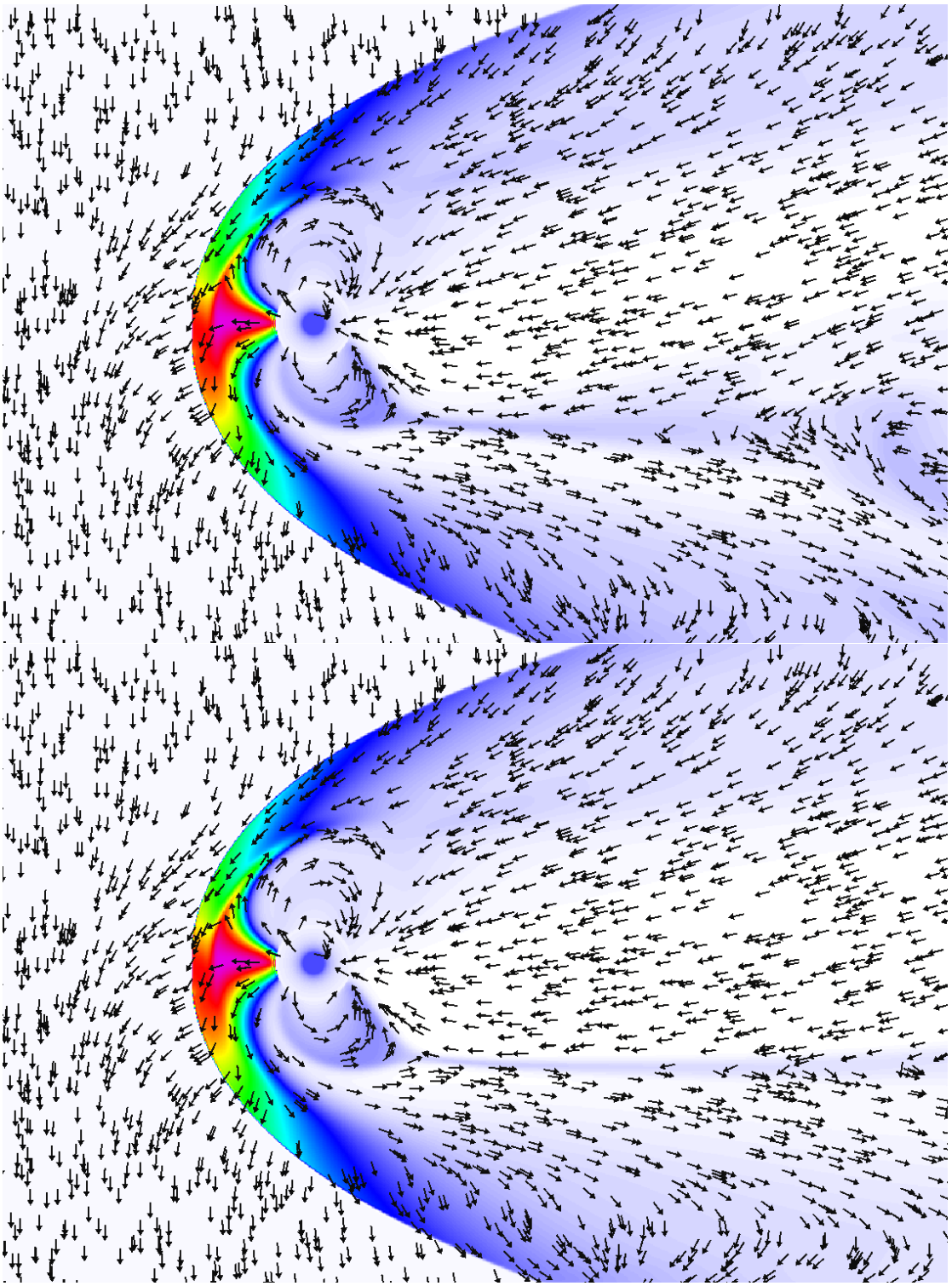

FIGURE 4.16: Results of the pressure evolution of the solar wind interaction with horizontal dipole field (3/3).

4.2.3 Discussion and Analysis

Phenomenon of solar wind interacting with the Earth's magnetosphere are reproduced with an approximated vertical dipole magnetic field. The evolution of the solar wind pushing the front side of the Earth's magnetic field to cause the bow shock, where the charged particles (ions) are deflected is reproduced in our simulation results. Other features such as the polar cusp, where the incoming solar wind particles can reach the atmosphere of the north pole and south pole (where the aurora appears), as well as the magnetosheath and the magnetopause are also reproduced in our simulation. Our simulation result show a very good symmetry to the $y = 0$ plane and $z = 0$ plane, which is in good agreement with the assumptions of [7]. These results show that the IMF has less influence on the magnetosphere since the magnetic field of the Earth is relatively strong.

Interesting findings occur in the tests of the inclined dipole. The results of the inclined dipole field are shown in Figures 4.11 to 4.13. The evolution of the magnetosphere shows a displacement of the vertical dipole field. The largest difference is the variation of the polar cusp. Because of the south pole is close to the direction that facing to the incoming solar wind, not only the pressure of the solar wind but also the size of the polar cusp is enlarged. The polar cusp becomes narrow and the pressure of the solar wind is weakened at the north pole. For the results of the horizontal dipole field, there is only one polar cusp (polar cusp of the south pole) that can be found clearly. The north pole is totally hidden inside the magnetosphere where it contains low-energy plasma. According to the magnetic field lines, charged particles are actually still able to get into the atmosphere at the north pole. Another discovery is the asymmetric pattern of the magnetosphere as well as a "wave-like" pattern of its evolution. This shows the affect of the IMF has been taken into account.

4.3 In-Situ Visualization

Performance tests achieve 4.38 TFLOPS in double precision for a computational domain of $1980 \times 1320 \times 1320$ using 216 GPUs. This shows that our large–scale global MHD simulations is very efficient. However, it is indispensable to record the simulation results in our research activities. For large-scale simulation, the amount of the data of a simulation result is very large. Recording such large amount of simulation data every time step for post-visualization is storage-demanding. In addition, writing files to disk is time consuming, resulting in a decrease in simulation performance. In our experiment, it took 25.6 seconds to record the 8 physics quantities of the MHD equations and 205.63

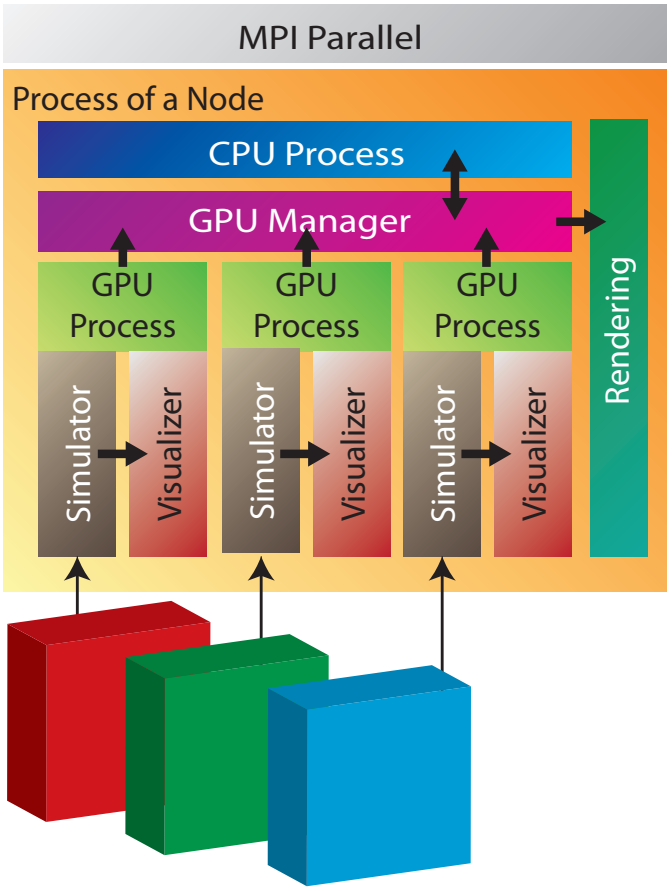

FIGURE 4.17: Real-time simulation and visualization for distributed multi-GPU system

GBytes was needed to store the data for one record. Therefore, in-situ visualization is significantly helpful in reducing disk storage and providing fast preview of the simulation results. In-situ visualization of large-scale simulation run on clusters has been a hot topic in the visualization research society. Serveral works have been proposed such as [86][87][88], but most of them are for CPU clusters. Burau et al. [41] also presented an in-situ visualization of their PIConGPU code, using the GPU for simulation, while using the CPU for in-situ visualization.

Our implementation of in-situ visualization of the large-scale global MHD simulations on distributed multi-GPU systems using GPU Direct-MPI hybrid framework will be explained in this section. When running a large-scale simulation using distributed multi-GPU systems, the simulation results of each partition are stored on the GRAM of the corresponding GPU. To efficiently perform visualization of the simulation data on each GRAM, a GPU based visualization program should also be run on each GPU. The visualization result of each partition is generated directly on the corresponding GPU, and will then be composited for rendering. A schematic of our framework is shown in Figure 4.17.

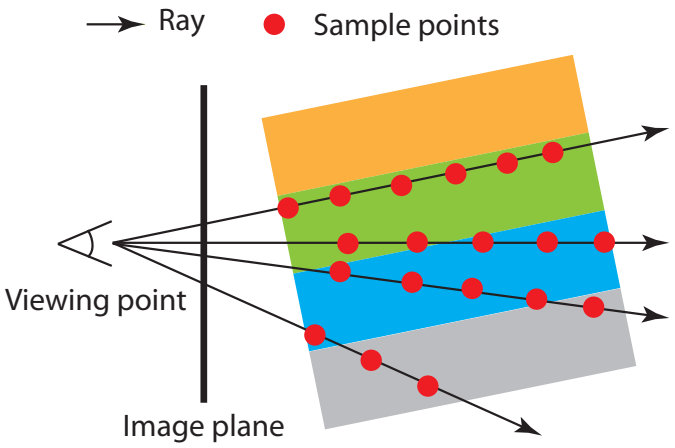

FIGURE 4.18: Volume ray casting for multiple partition volumes

For real-time direct volume visualization, volume ray casting [68] on multiple GPUs is implemented and used. The volume ray casting is one of the widely used direct volume rendering (DVR) algorithms for visualizing volume data. The color of a pixel on the image plane is calculated as the projection of the sample points along a ray shot through the volume. Volume ray casting algorithm calculates the accumulation of all the sample points of the data volume lying on a ray. Equation 4.7 of the ray casting algorithm allows us to calculate the result of each partition first and then combine them together as shown in Figures 4.18 and 4.19 . Blending the results of two partitions has no difference from blending the RGBA value of two sample points. What we need to do is to calculate the α factor correctly. By extending Equation 4.7 to Equation 4.8, where $C_{current}$, $\alpha_{current}$ represent the current accumulated Color and α, C_{new}, α_{new} represent the Color and α of a new sample point, $C_{partition_n}$ and $\alpha_{partition_n}$ represent the accumulated Color and α of a partition. For each partition n, $(C_{partition_n}, \alpha_{partition_n})$ is calculated by accumulation of its sample points. After that, the results of all partitions $(C_{partition_n}, \alpha_{partition_n})$ lying on the same ray will be accumulated to generate the final (C, α) of a pixel of the image plane.

$$\begin{aligned} C &= C_{current} + (1.0 - \alpha_{current}) C_{new} \alpha_{new}, \\ \alpha &= \alpha_{current} + (1.0 - \alpha_{current}) \alpha_{new}. \end{aligned} \tag{4.7}$$

$$\begin{aligned} C_{partition_n} &= C_{current} + (1.0 - \alpha_{current}) C_{new} \alpha_{new}, \\ \alpha_{partition_n} &= \alpha_{current} + (1.0 - \alpha_{current}) \alpha_{new}, \\ C &= C_{partition_1} + (1.0 - \alpha_{partition_1}) C_{partition_2} \alpha_{partition_2}, \\ \alpha &= \alpha_{partition_1} + (1.0 - \alpha_{partition_1}) \alpha_{partition_2}. \end{aligned} \tag{4.8}$$

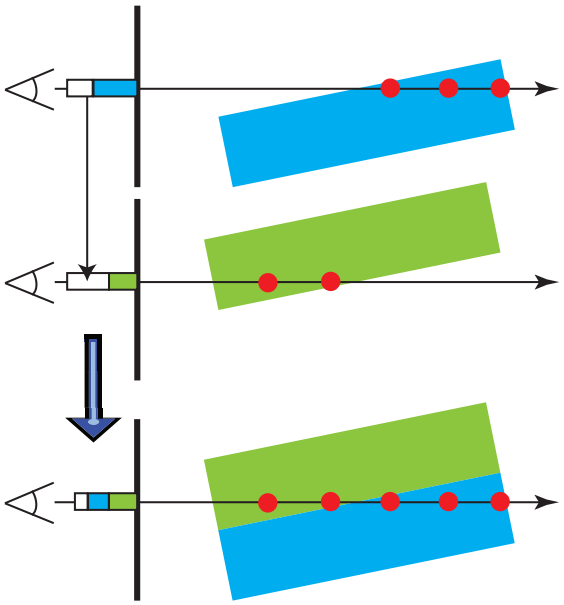

FIGURE 4.19: Composition of the volume ray casting for multiple partition volumes

One important issue is that the coordinate system of the viewing volume of each partition must be aligned with the cell face of the boundary grids (not the halo grids). This issue is seldom being considered in volume visualization of a single volume. Since the partitions won't be composed to the whole dataset, if the coordinates between each partition do not align, overlapped grids or a gap will appear in the composed results (see Figure 4.20). On the other hand, when processing volume visualization on GPUs, the dataset is stored as a 3D texture. It is common to use normalized coordinate for the texels. However, using a normalized coordinate system for each partition will make the composition process much more complicated. More transformation and scaling of the coordinate system will be needed. In addition, the precision(machine error) of these calculations may cause two partition to become misaligned. Therefore, we suggest to use the original coordinates as the computation domain of the simulation. Choosing between normailized or non-normailized coordinate systems when creating the 3D texture is very simple. It can be done by changing a parameter in the CUDA APIs.

Blending order is another key point of combining the visualization results of partitions in DVR using multiple GPUs. Blending order must be the same as the accumulation order of the sample points of volume ray casting. For example, if the ray casting is front-to-back, then the blending order of the partitions must also be front-to-back. As shown in Figure 4.21, the order of the partition will be different depending on the position of the viewer. Thus, the direction and the position of the viewing points have to be tested and broadcasted to all computing node before combining the visualization results of the partitions. Figure 4.21 shows an example of composition in 1D. For 3D composition, we

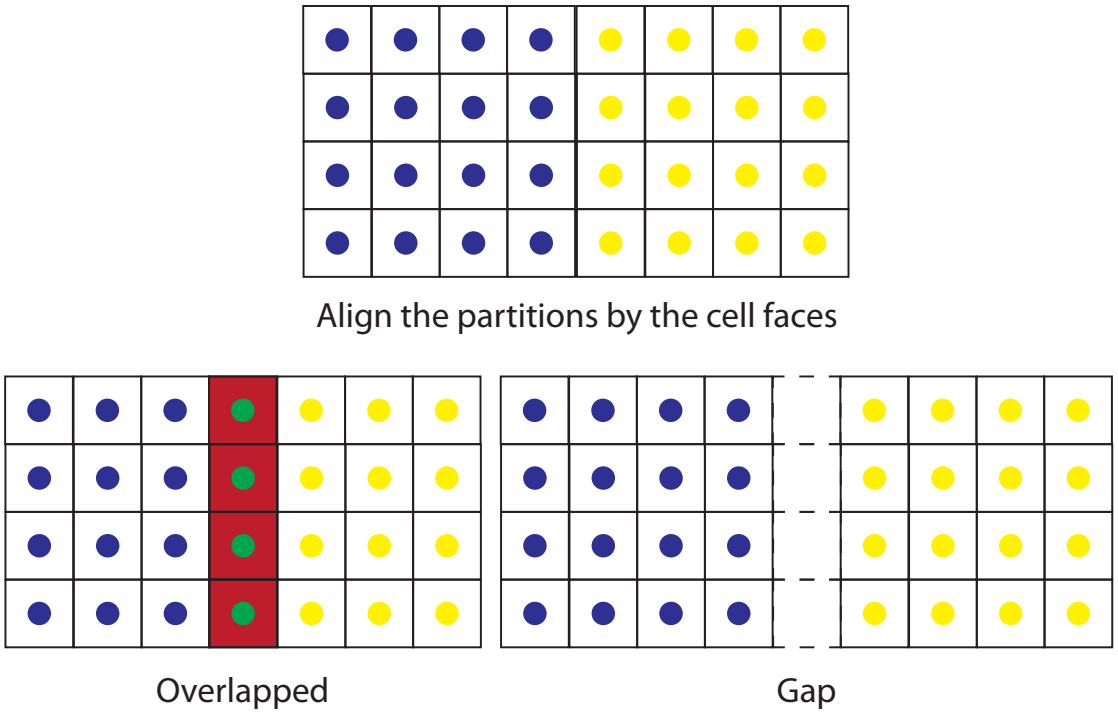

FIGURE 4.20: Align the partition by cell face to prevent overlapped grid points or a gap.

can simply blend the results in each direction one by one. For example, first blending in the z-direction and then the y-direction, and finally the the x-direction. The information of the viewing point and the decomposition of the computational domain of the simulation can be used to identify the correct blending order.

A reduction algorithm is used for combining the visualization results (see Figure 4.22). Based on our GPU Direct-MPI hybrid framework, the data communication is a 2-layer model. Thus, the composition of the visualization results are also done in two steps. The visualization results of multiple GPUs of each computing node are being composited via GPU Direct peer-to-peer communication first. Then, the visualization results of the computing nodes will be composited. For example, for a computing node that contains four GPUs, which are named GPU_0, GPU_1, GPU_2 and GPU_3, the reduction process will composite the results between GPU_0 and GPU_1, GPU_2 and GPU_3, and then composite the results between GPU_0 and GPU_2, as shown in Figure 4.22. After that, the same reduction process will be performed between the computing nodes. An example of composing the final visualization results from 4 partitions of our MHD simulation is shown in Figure 4.23. Rather than gathering the data to a single node for visualization, our approach copies the visualization results to another GPU/computing node. As a result, it has less overhead of data copy and requires less memory than gathering the data of the whole computational domain onto a single GPU for visualization. Furthermore,

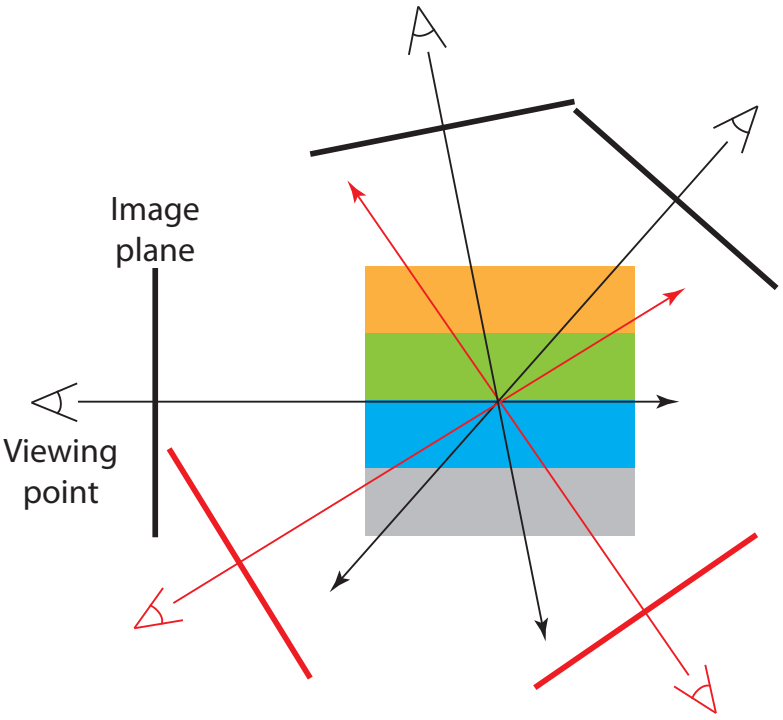

FIGURE 4.21: Blending order for combining the visualization results (Black : forward blending. Red : reverse blending).

every GPU of every computing node invokes not only the simulation process but also the visualization process, resulting in better load balancing than using specific visualization nodes.

For composing the visualization results, one important issue is that the coordinate system of the viewing volume of each partition must be aligned with the cell face of the boundary grids. Otherwise, overlapped grids of a gap will appear when combining the visualization results.

Large-scale global MHD simulations were run using the TSUBAME 2.5 GPU-rich supercomputer of the Tokyo Institute of Technology. All test results were computed in double precision. Our global MHD simulation performs 4.38 TFLOPS with the resolution of $1980 \times 1320 \times 1320$ domain, using 216 Kepler (K20X) GPUs of TSUBAME 2.5. Figure 4.24 shows the in-situ visualization results of the solar wind-Earth's magnetosphere interaction with lower resolutions ($810 \times 540 \times 540$) of the simulation domain. The direct volume visualization results of the partition domains was generated and composited using multiple GPU as mentioned in the above section. Rendering in real-time to the screen via OpenGL was done simultaneously with the simulation using multiple GPUs on TSUBAME 2.5. The evolution of the simulation as well as the appearance of the bow shock can be visualized in on the fly for each simulation step. The FPS is bottlenecked by the overheads of synchronization, image composition, and the data communication of

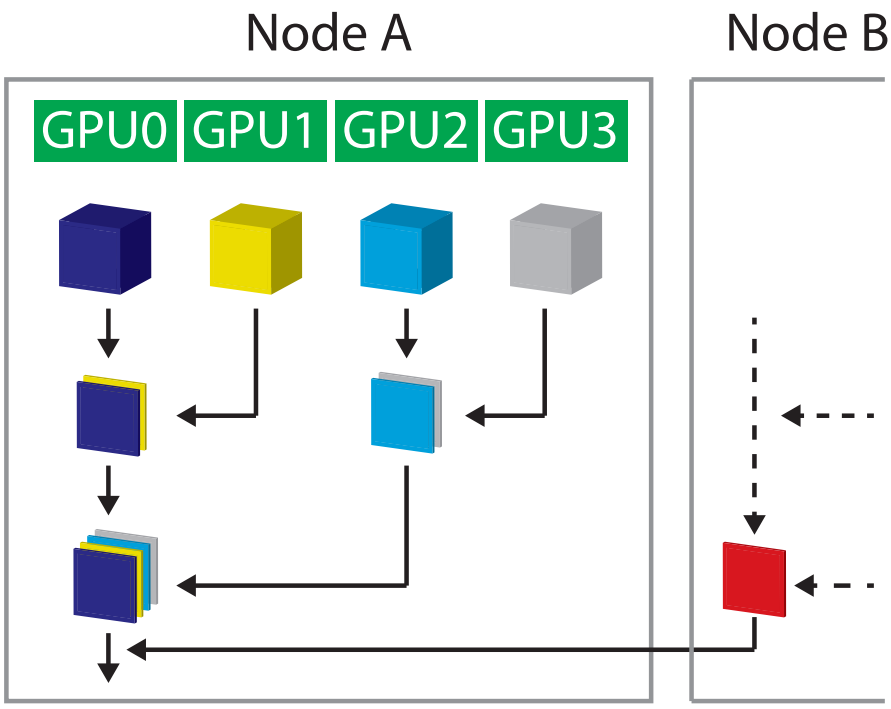

FIGURE 4.22: Composite the visualization results of partition domain by reduction

TABLE 4.1: Performance of the real-time global MHD simulation and visualization on TSUBAME 2.5

Domain resolution	Number of GPUs	Step per Second (=FPS)	ms/step
180 × 120 × 120	1	7.4	135.14
180 × 120 × 120	3	13.2	75.76
270 × 180 × 180	3	6.9	144.93
270 × 180 × 180	6	9.9	101.01
540 × 360 × 360	24	5.7	175.44
540 × 360 × 360	81	8.2	121.95
810 × 540 × 540	81	5.0	200.00

remote rendering (OpenGL via ssh X11 forwarding). Nevertheless, 5 to 13.2 FPS can be achieved with different resolution and number of GPUs. The results are shown in Table 4.1. The results show the steps per second, which is equal to FPS since visualization is done on the fly for every simulation step. In our analysis, most of the computational time is spent on the simulation itself. Therefore, user interaction such as changing the viewing angle does not show significant difference in the frame rate.

4.4 Summary

In this section, large-scale global MHD simulations for solar wind interaction with the Earth's magnetosphere using distributed multi-GPU system developed using our GPU

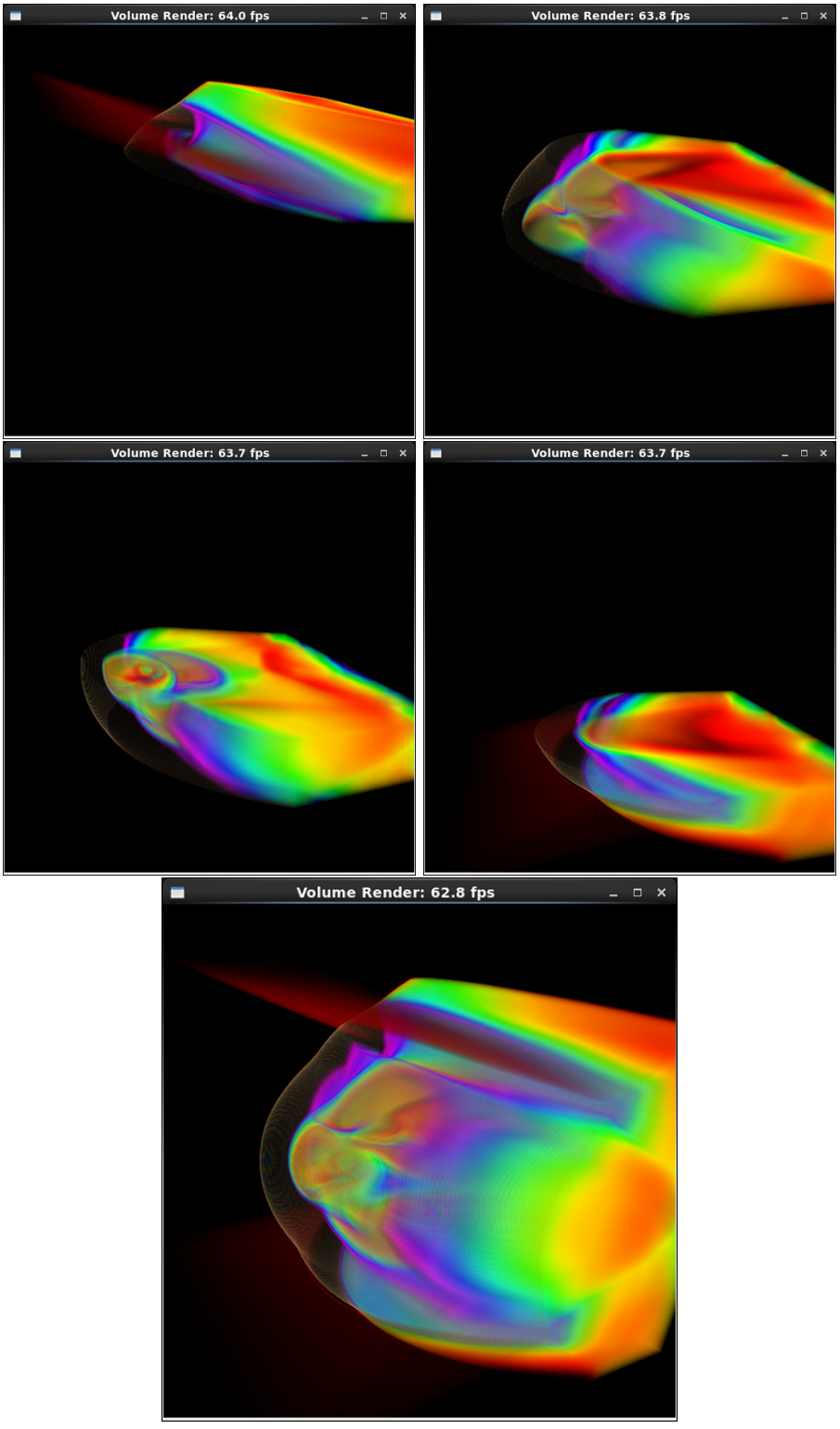

FIGURE 4.23: Real-time rendering results of the composited final image (bottom) from the visualization results of 4 partitions (top).

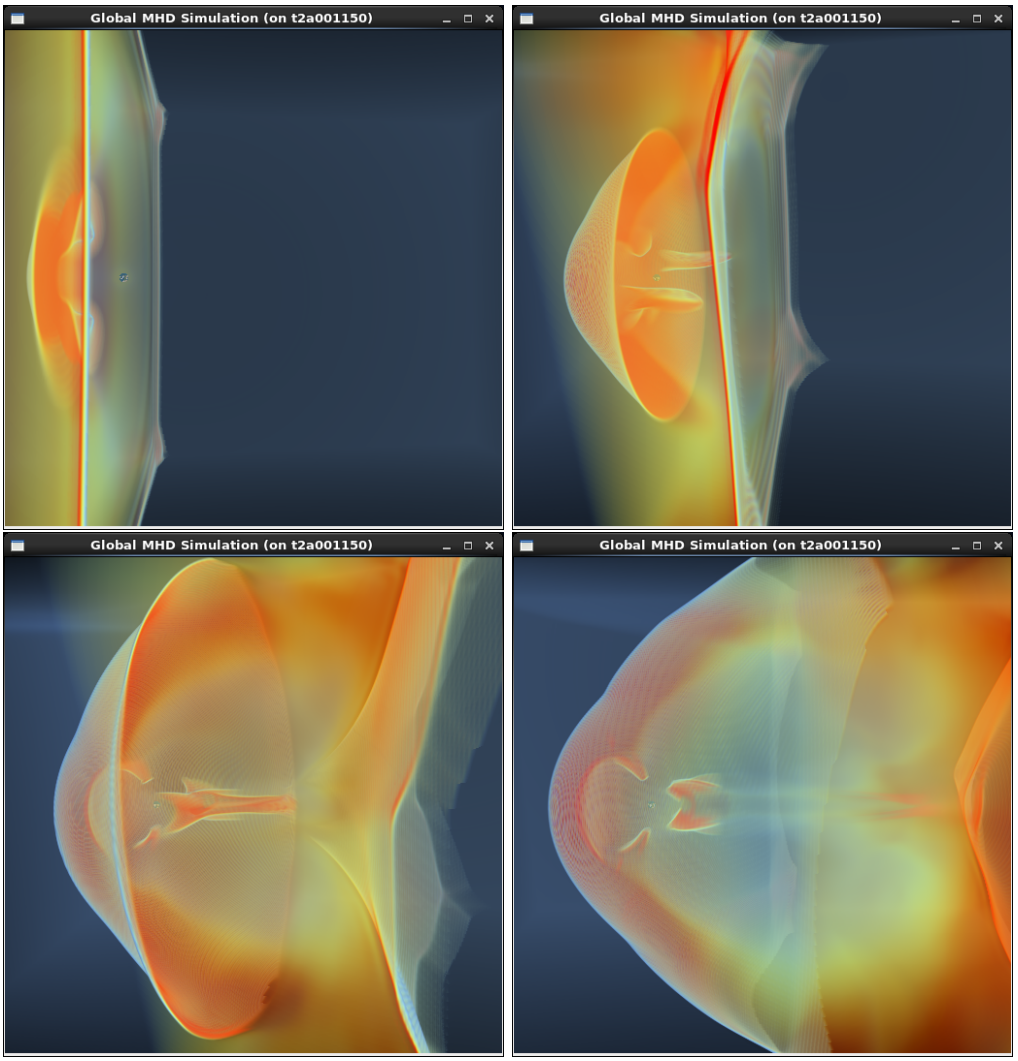

FIGURE 4.24: Real-time visualization of the solar wind-Earth's magnetosphere interaction

Direct-MPI hybrid framework is presented. By utilizing the high floating-point computational power of GPU to accelerate the calculation, the performance reached 4.38 TFLOPS in double precision for simulations with a resolution of $1980 \times 1320 \times 1320$ domain using 216 GPUs of TSUBAME 2.5. Simulations for the geomagnetic reversal were also carried out. A unit quaternion is used for the rotation of the dipole field. Compared to the vertical dipole field, the asymmetry pattern of the magnetosphere caused by the inclination of the dipole field as well as the IMF is found. On the other hand, the slow I/O time for recording the large amount data from the large-scale simulation is pointed out. To solve this problem, an in-situ visualization on distributed multi-GPU systems is presented. Simulation and visualization are processed using multiple GPUs simultaneously. By minimizing the overheads of copying and gathering the data, only the visualization results of each partition needed to be composited for rendering. Experimental results of our global MHD simulation and visualization running 81 GPUs

achieves up to 8.2 FPS and 5.0 FPS for the simulations of solar wind magnetosphere interaction for the $540 \times 360 \times 360$ and $810 \times 540 \times 540$ domain, respectively.

Chapter 5

Advanced Simulation for Solar Wind – Earth's Magnetosphere Interaction

5.1 Computational Resource Issues

The resolution or size of the calculation domain of the numerical simulation is limited by the memory of the system. As we discussed in previous sections, the computational complexity of global MHD simulation is high. Many new techniques in HPC has been proposed to speedup the process. However, 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 section, 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. In addition, an efficient adaptive mesh refinement (AMR) on multi-GPU systems is developed using our approach. As a result, we enlarge the simulation domain of the simulation to reproduce the full structure of the magnetosphere. Restrictions on the boundary condition of the solar wind interaction with the magnetosphere is also clarified. Improvements of the boundary condition as well as new findings of the simulation results are also presented.

5.2 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 how a partial differences equation (PDE) and see how it relates to a simulation program. As an example, a PDE is shown in Figure 5.1, where the following descriptions are given:

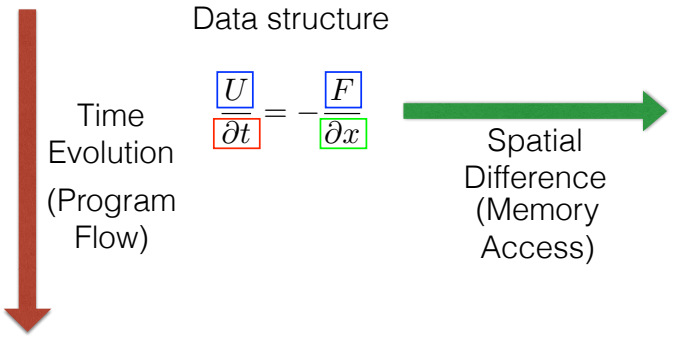

FIGURE 5.1: The perspective of how a PDE relates to the components of a program

1. The physics quantities U and F are reflect to data storage and data structure (mesh or particles)
2. ∂t is related to the programming flow – the sequence of the simulation code
3. ∂x is related to the data access between the elements – the stencil computation.

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

1. The whole mesh is decomposed into a fixed number of partitions (called blocks hereafter).
2. Each block is a subset of the whole computational domain, where the resolution of each block may vary.
3. Each block has only one neighbour in each direction $(x+, x-, y+, y-, z+, z-)$.
4. A geometry map is used to link all the blocks, and also for finding the adjacent block (in Figure 5.2).

Multi-stream task-parallel and adaptive mesh refinement (AMR) based on this data structure will be introduced in the following sections.

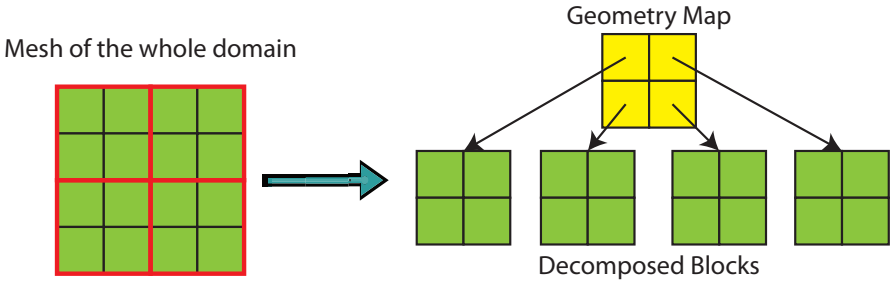

FIGURE 5.2: The block-based structured mesh

5.2.1 Multi-stream Task Parallel on GPU

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

1. Calculate F^t from the governing equations using U^t
2. Evolve U^t to U^{t+dt} (for example, using the Runge-Kutta method)
3. Addition processes such as limiters, smoothing etc. .
4. Repeat from step 1 until the simulation ends.

In this 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 or other multiple steps scheme 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 [82]. In our block-based structure, a block is a unit to be processed. Therefore, the storage of the intermediate 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. In our tests, memory usage of our MHD simulation with 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 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 (needed by the scheme) and 0.056 GB for the workspace, respectively. Although 0.1 GB is increased of the data communication (the halo) between the blocks for the physics quantities and the half grid data, a huge number of memory of 2.394 GB is saved for the workspace. In total, our block-based structure save 2.194 GB. As shown in Figure 5.3, 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 GPU. Moreover, it is possible to launch multiple tasks with different calculation kernels. For example, as shown in Figure 5.4, 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.

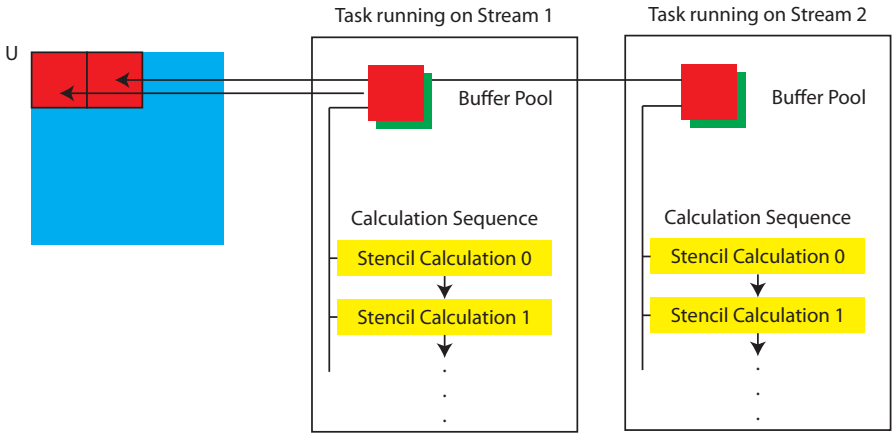

FIGURE 5.3: Multi-stream task parallel on GPU with the block-based structure(same task)

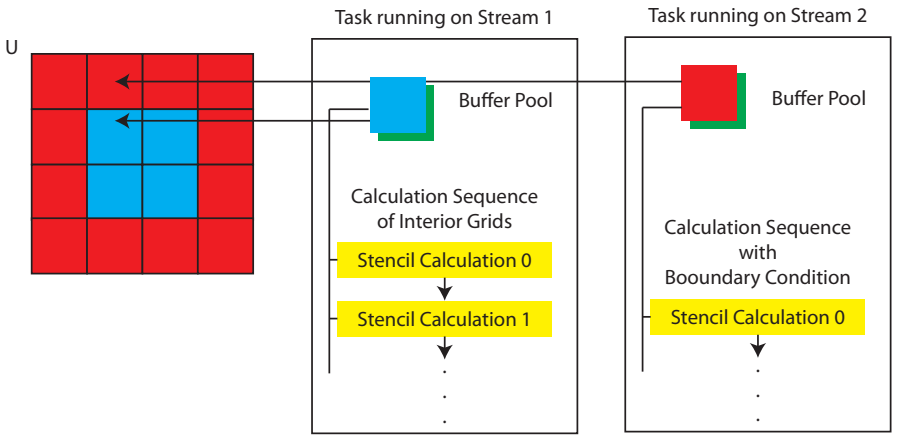

FIGURE 5.4: Multi-stream task parallel on GPU with the block-based structure(different tasks)

5.2.2 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 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 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 Figure 5.5, 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.

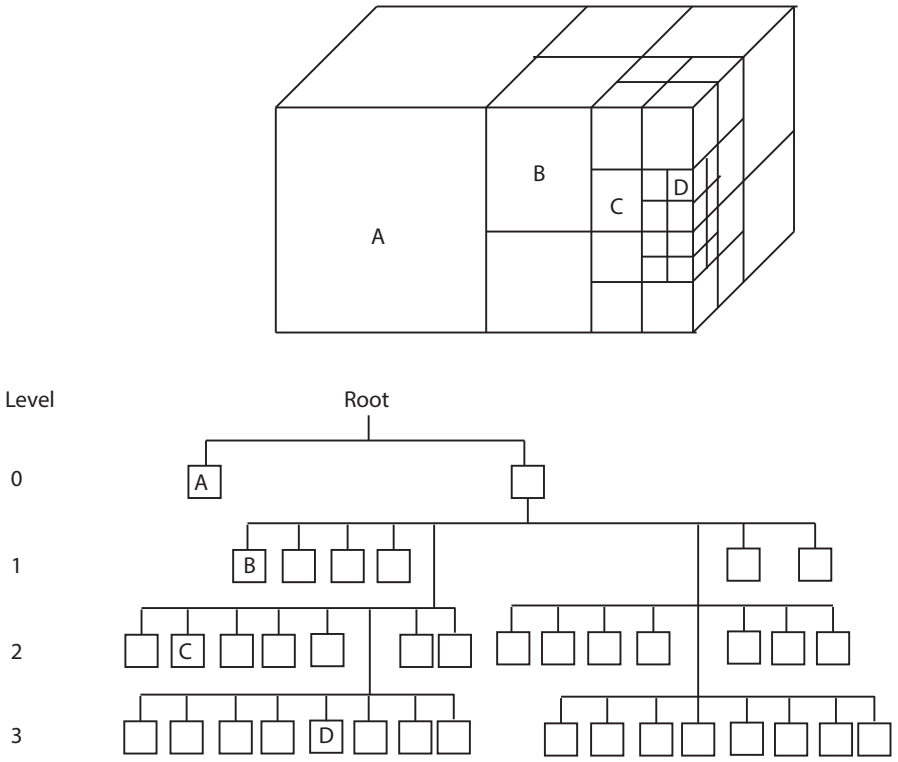

FIGURE 5.5: Tree data structure for AMR.

Adaptive mesh refinement (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. 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 [89] use the CPU to help data management and communication. It is easy to perform AMR by changing the resolution of each block of our block-base structure (see Figure 5.6). 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 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 adjacent grid (as shown in Figure 5.8). Therefore, to achieve further speedup, the following strategy is applied:

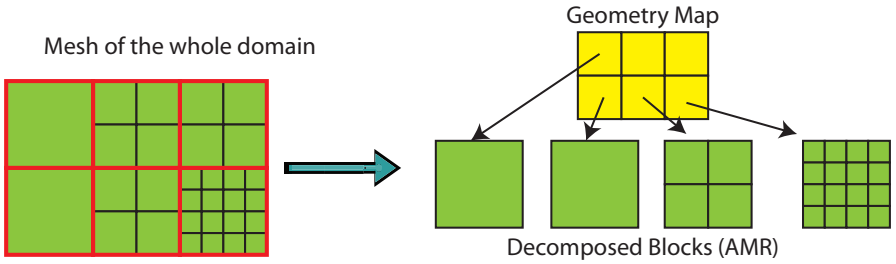

FIGURE 5.6: AMR of the block-based structure.

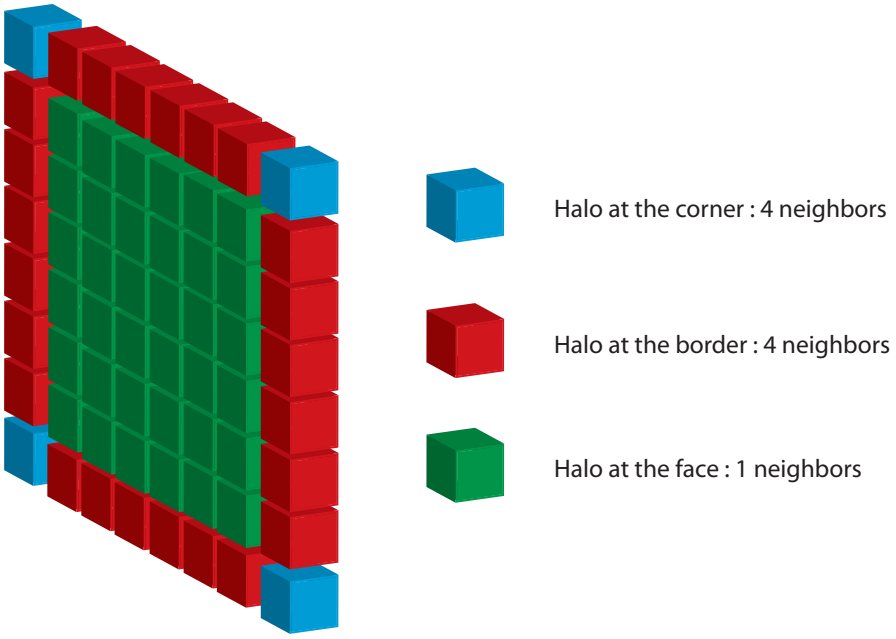

FIGURE 5.7: The 9 neighbours for updating one slice of the halo (in one direction) of a 3D mesh-based simulation.

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, neighbour 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.

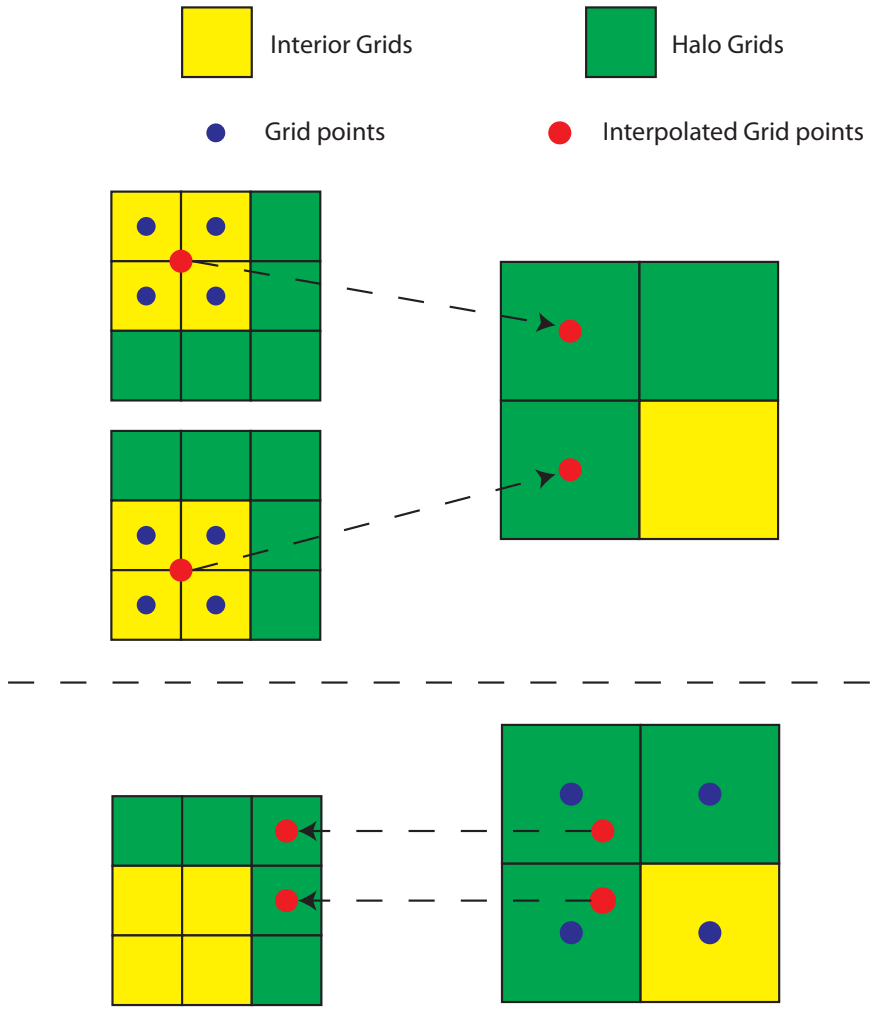

FIGURE 5.8: The interpolation of grid points between different level

Refine the resolution (changing the level) of a block is time consuming. According to the modified leapfrog scheme (Appendix B, Figures B.1 and B.2), 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.

The dipole field $\mathbf{B}_d$ which is invoked in the simulation but never changed by the simulation, is stored in texture memory. Texture memory is a special feature of GPU which not only provids fast read with a specific texture cache, but also perform fast linear interpolation. Even though it only supports single precision, the accuracy is enough for the constant value $\mathbf{B}_d$.

5.3 Restriction and Improvement for Simulating the Whole Bow Shock

Due to the lack of computational power, the simulation domain of the solar wind and Earth's magnetosphere interaction introduced by Ogino *et al* [7, 23, 24] was 1/4 the size of the simulation domain of the model $(x, y, z) = (-60R_e, 0R_e, 0R_e) \sim (30R_e, 30R_e, 30R_e)$. Symmetry (mirror) boundary condition was applied to the boundary of $y = 0$ and $z = 0$. After the upgrade of the computer hardware, it was extended to the full domain $(x, y, z) = (-60R_e, -30R_e, -30R_e) \sim (30R_e, 30R_e, 30R_e)$ and has been applied to solar wind interaction with the Saturn's magnetosphere. However, 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 figure 5.9

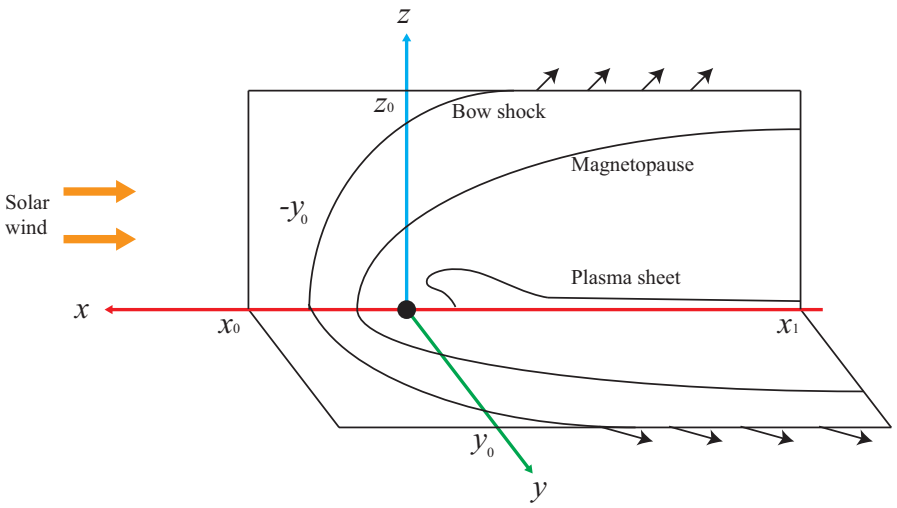

FIGURE 5.9: The boundary condition at 45 degrees to the x-axis

The reason for this boundary condition setting is the assumption of making the magnetic field align with the blow shock. Otherwise, an erroneous reflection will appear during the simulation as shown in Figure 5.10:

The 45 degrees shear boundary condition also lead to difficulties for the data updates at the halo region. There are 2 workflows to update all the halo grids. As shown in the left figure in Figure 5.11 :

1. Data communication in x-direction is processed.
2. Update the boundary condition in y-direction.
3. Data communication in x-direction is processed to update the halo grid at the corner

An alternative workflow is shown in the the right figure in Figure 5.11:

1. Update with the boundary condition in y-direction.
2. Data communication in x-direction is process.
3. Update with the boundary condition in y-direction.

Therefore, whether we copy the data first or process of the boundary condition first, an additional second copy(boundary condition process) is always needed to update all the halos.

On the other hand, it is found that there is minimum requirement of $dx \leq \frac{R_e}{2}$ for the resolution for the solar wind and Earth's magnetosphere interaction. This leads to a minimum requirement for the memory for a given size of simulation domain. To assess the problem and the affected region, we use the block-based structure we introduced in Section 5.2 and decompose the domain in several partitions (blocks) and analyze the simulation. We found that the erroneous reflection did not occur if a normal Neumann boundary condition is applied to the region where this boundary is outside of the bow shock. This evidently shows that it is not necessary to use the 45 degrees shear Neumann boundary condition if all the y and z boundaries are outside of the bow shock. Use of the block-base structure allows us to save the memory usage and extend the simulation domain from $(x, y, z) = (-60R_e, -30R_e, -30R_e) \sim (30R_e, 30R_e, 30R_e)$ to $(x, y, z) = (-70R_e, -70R_e, -70R_e) \sim (70R_e, 70R_e, 70R_e)$ which is large enough to cover the boundary of the bow shock as well as the whole magnetosphere in the y and z direction. Therefore, the Neumann boundary condition is applied to the y and z boundary without the erroneous reflection as shown in Figure 5.10.

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 Figure 5.13, the blocks of the large gradient of the pressure field are refined to the highest level. The neighboring blocks in y-direction(z-direction, also) are also refined as the highest 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 GPUs

and speedup the simulation from 1070.19 ms/step to 938.406 ms/step. However, the process of refining the blocks requires additional 7304.87 ms to process the refinement.

Simulation results of the whole magnetosphere with slow solar wind (400 km/s) are presented in Figures 5.14 and 5.15. Simulation results with fast solar wind (750 km/s) are given in Figures 5.16 and 5.17. Beside the obvious difference in the strength of the pressure, comparison of the results between slow and fast solar wind reveals that the size of the bow shock, as well as the size of the magnetosheath and the position of the magnetopause is different in the z-direction. However, there is no difference in the y-direction. Comparison of figures are shown in Figures 5.18 to 5.20 to see the difference clearly.

The Moon orbits the Earth with a mean radius of 385×10^3 km $\simeq 60.3R_e$ with mean inclination of 5.145 degree. The inclination of the Moon's orbit means that the Moon doesn't pass the plasma sheet but the Northern tail lobe and it across the magnetosheath and the magnetotail. Since we enlarged the simulation domain to $(x, y, z) = (-70R_e, -70R_e, -70R_e) \sim (70R_e, 70R_e, 70R_e)$, the whole trajectory of the Moon's orbit is included in our simulation. The clipping plane of the Moon's orbit is illustrated in Figure 5.21. Analysis of our simulation results for the Moon's orbit are shown in Figure 5.22. Although the size of bow shock is almost the same, the different sizes of the magnetosheath as well as the magnetotail are found. These results show that under the different solar wind speed, the period of the Moon exposed directly to the sun is similar. However, the period of staying in the magnetosheath, where the density of the particles is lower than that outside of the bow shock, is quite different. The Moon doesn't have a magnetic field to block the incoming charged particles. Understanding the period of the Moon contacting with different density of charged particles is very important.

5.4 Summary

In this chapter, we deal with the limitation of the size of calculation domain. We found the restriction of the current global MHD simulation of the boundary condition at the y and z boundary that not only affects to the simulation results, but also the performance. 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. The performance of AMR is actually based on the problem. In our 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 total memory usage from 4.670 GB to 3.504 GB for each GPUs and speedup the simulation from 1070.19 ms/step

to 938.406 ms/step. However, if mesh refinement is needed, the simulation AMR requires additional 7304.87 ms to process the refinement. We enlarge the simulation domain, which enables our simulation to contain the whole bow shock. Therefore, the specific 45 degree shear Neumann boundary condition can be replaced with a simple Neumann boundary condition. We also simulated and compared the results of slow and fast solar wind interaction with the Earth's magnetosphere. Different height (size in z direction) but similar width (size in y direction) of the bow shock as well as magnetosheath is found. The magnetosphere is taller when solar wind speed is low. In addition, the region of the Moon's orbit across the magnetosphere is also examined. It is found that, the period of the Moon exposed directly to the sun is almost the same but the period at which it stays within the magnetosheath is different. The density of the charged particles within the magnetosheath is lower than the region found beyond the bow shock, but greater than within the magnetopause. Thus, the period and the amount of the charged particles hitting the lunar surface is changed when the solar wind speed is changed.

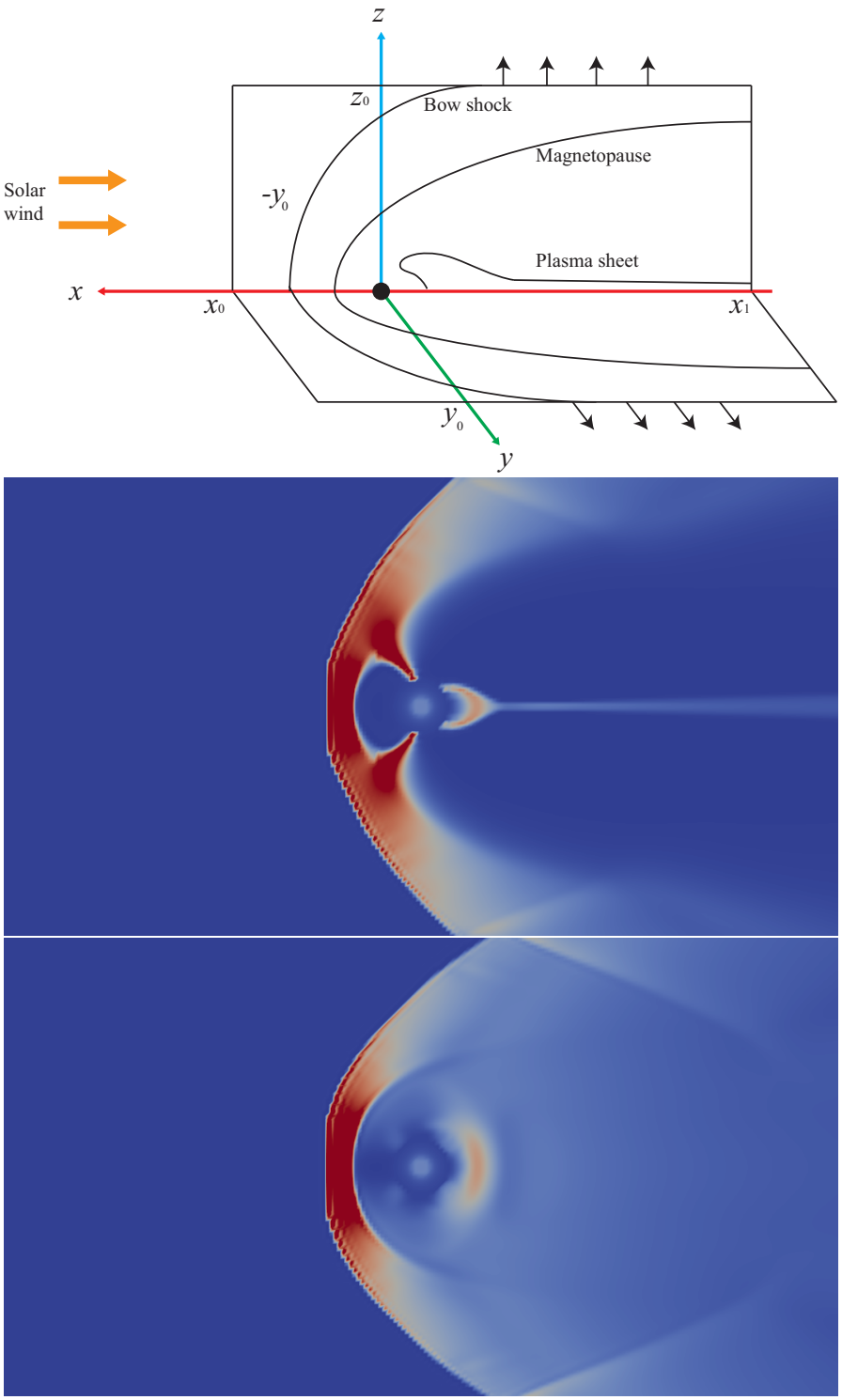

FIGURE 5.10: A Neumann boundary condition is applied to the y and z boundary and the erroneous reflection

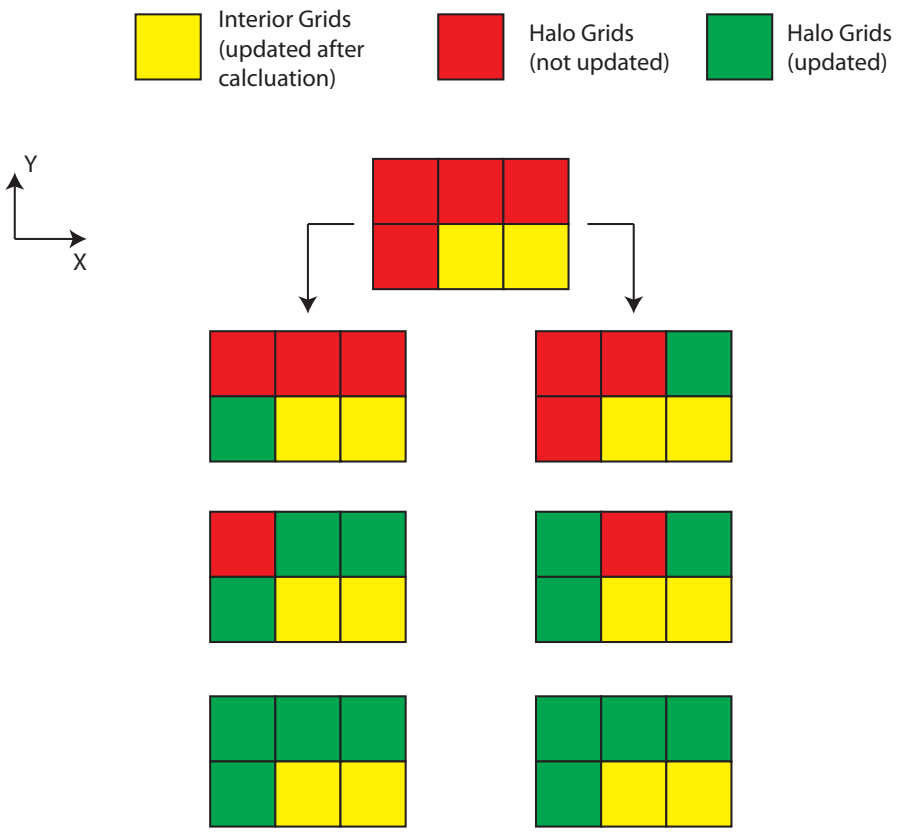

FIGURE 5.11: Two times of data exchanges or boundary condition processes are needed to updated all the halo

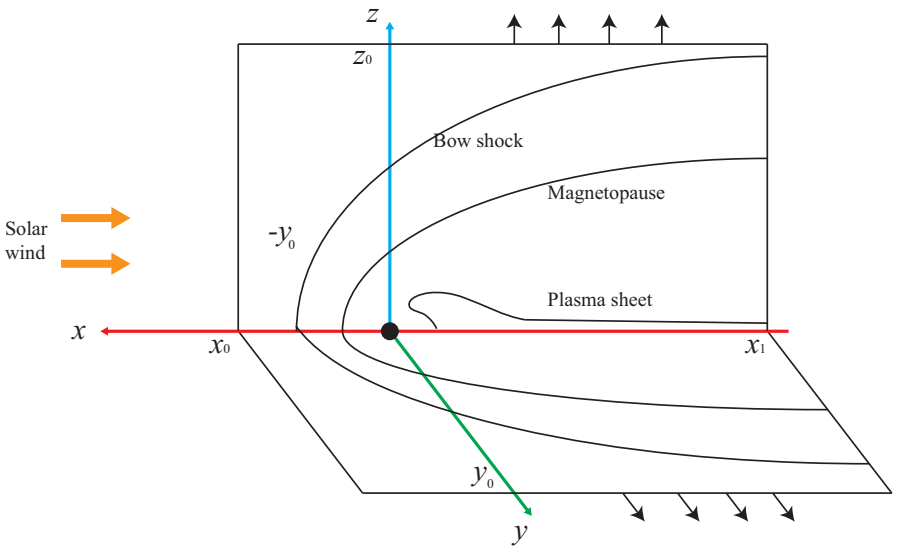

FIGURE 5.12: Enlarged simulation domain which contains the whole magnetosphere with Neumann boundary condition.

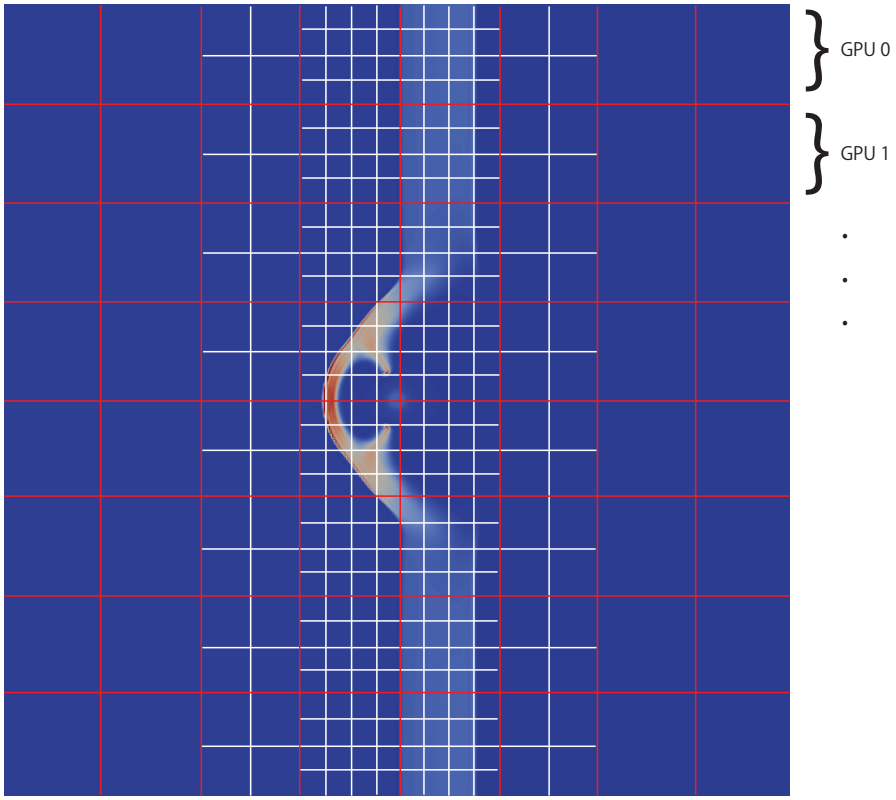

FIGURE 5.13: Image illustrates the blocks with different level (Note that it is not the real resolution). Refining the mesh along x-direction and distribute to each GPUs by the other direction for less communication overhead and well load balancing. Red lines show the partition of the blocks.

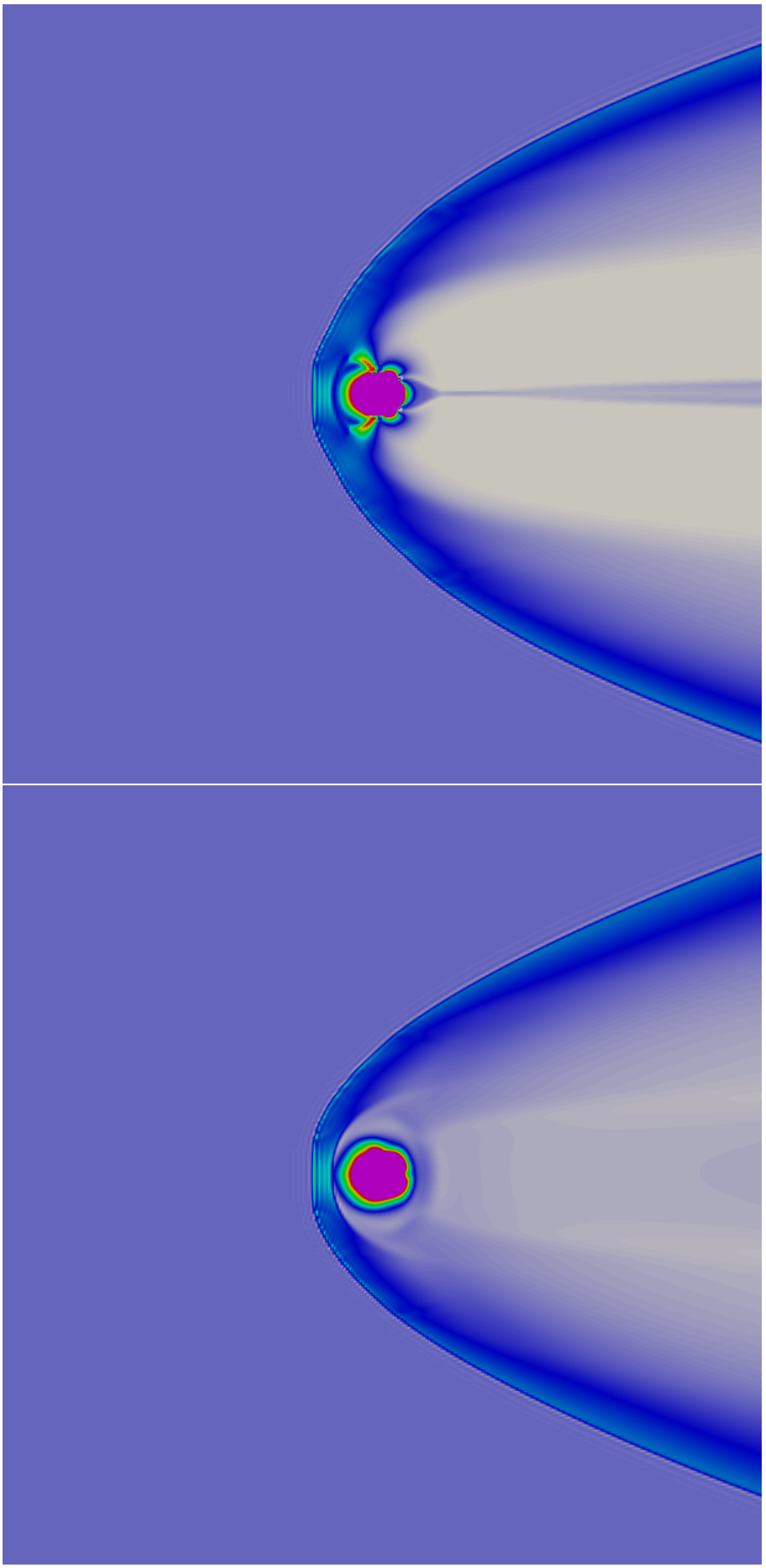

FIGURE 5.14: Simulation results (density : ρ) of the full structure of the Earth's magnetosphere with slow solar wind (400 km/s). Top : X-Z plane. Bottom : X-Y plane.

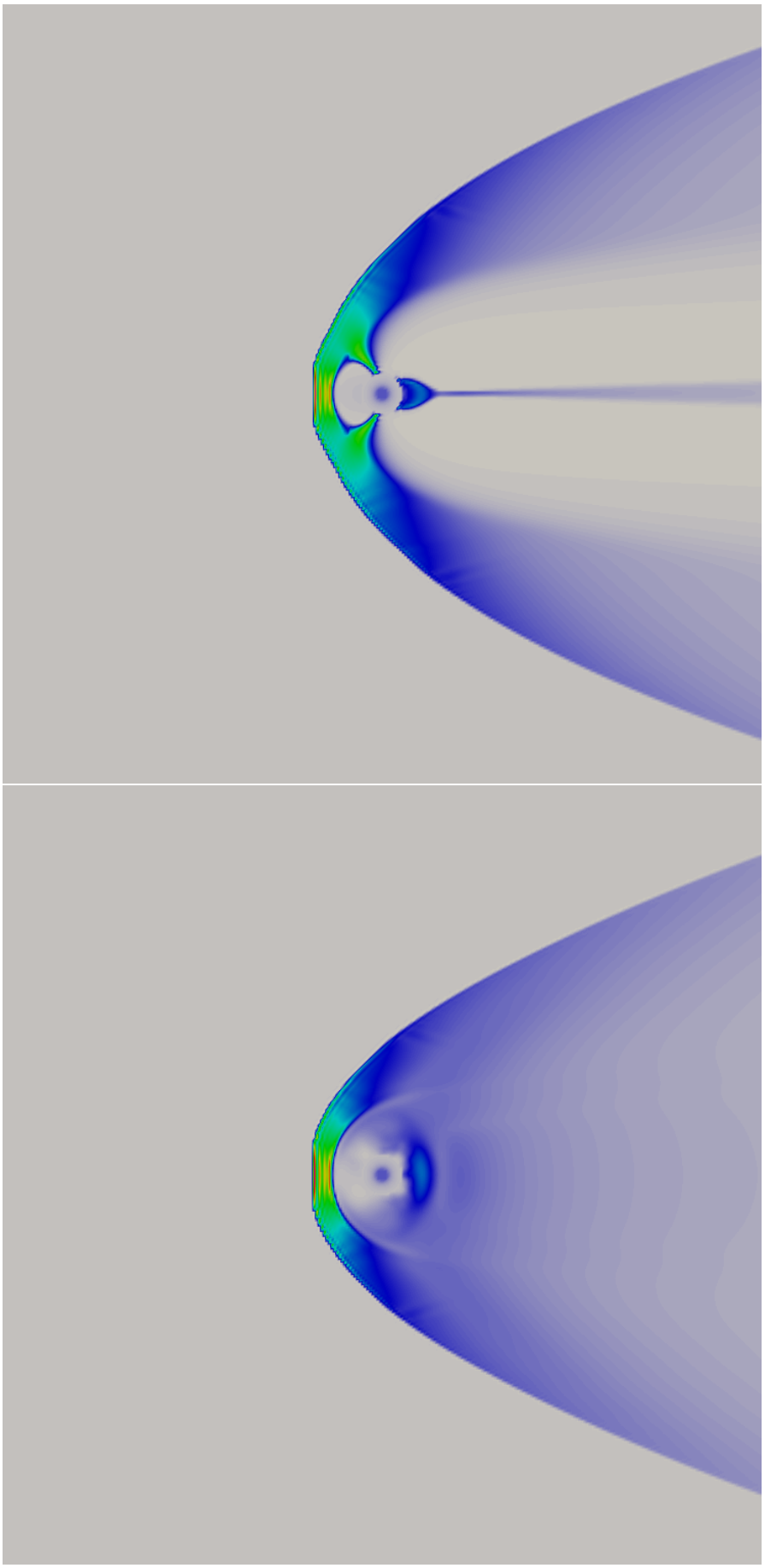

FIGURE 5.15: Simulation results (total pressure : p) of the full structure of the Earth's magnetosphere with slow solar wind (400 km/s). Top : X-Z plane. Bottom : X-Y plane.

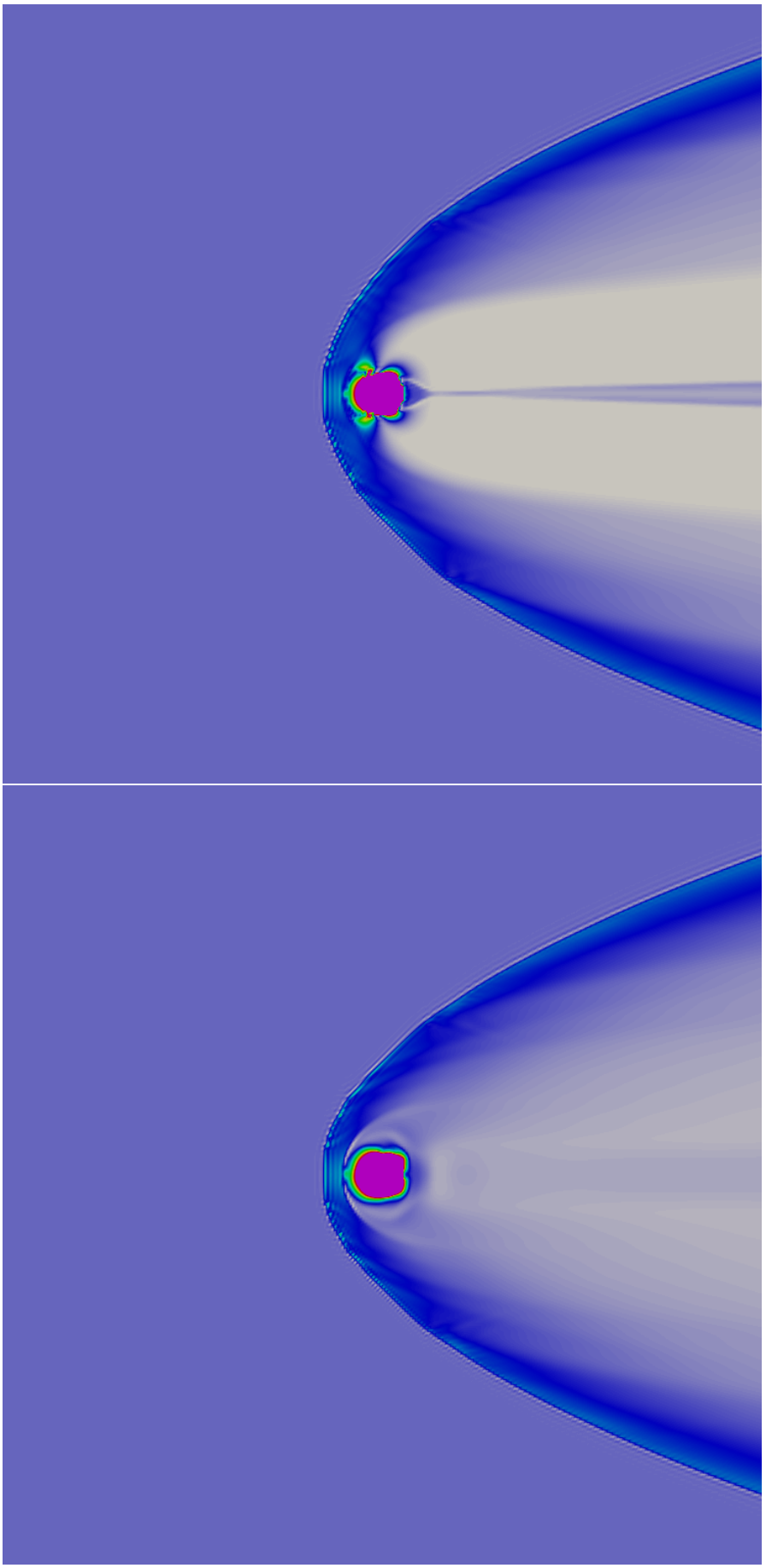

FIGURE 5.16: Simulation results (density : ρ) of the full structure of the Earth's magnetosphere with fast solar wind (750 km/s). Top : X-Z plane. Bottom : X-Y plane.

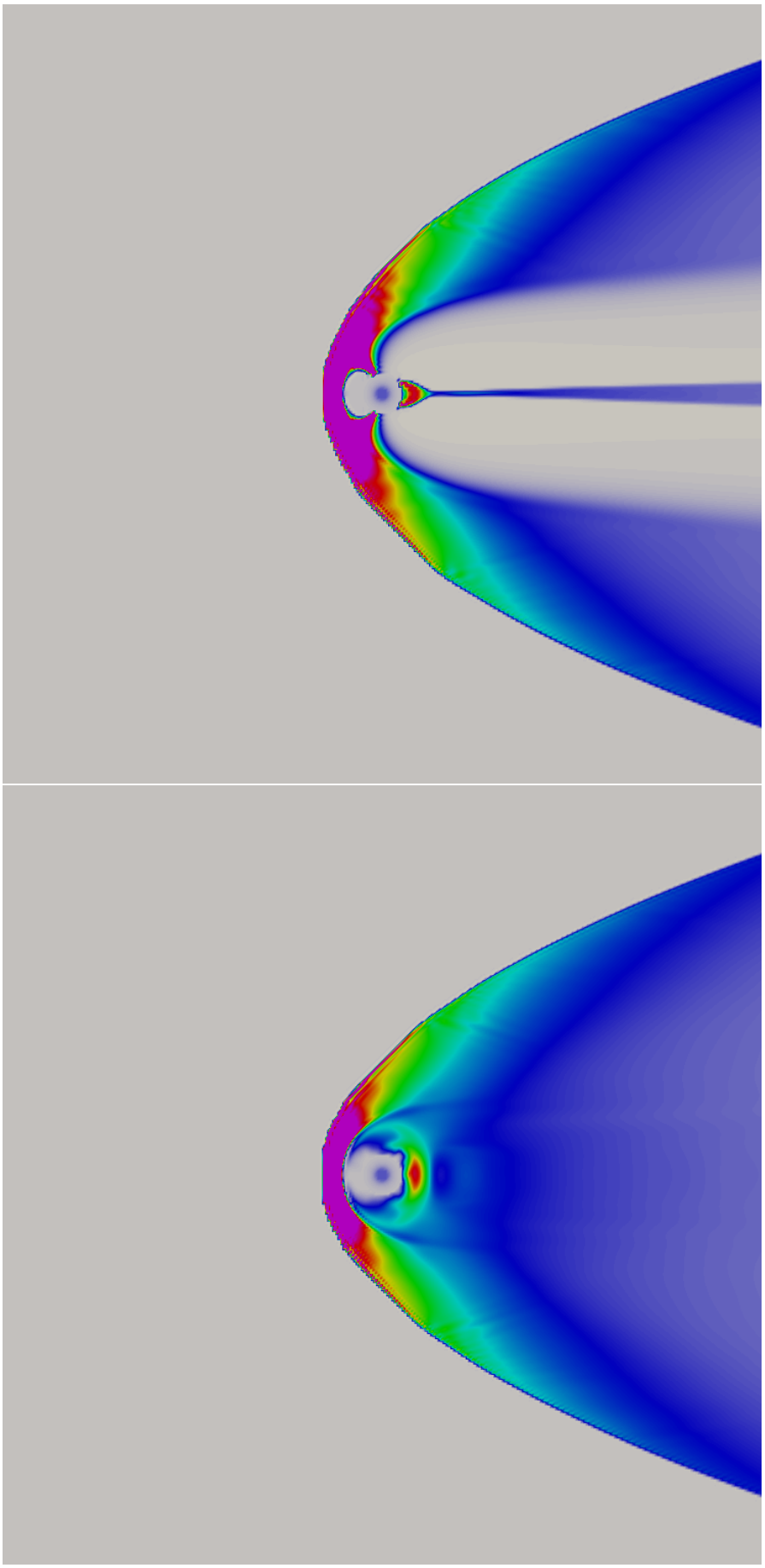

FIGURE 5.17: Simulation results (total pressure : p) of the full structure of the Earth's magnetosphere with fast solar wind (750 km/s). Top : X-Z plane. Bottom : X-Y plane.

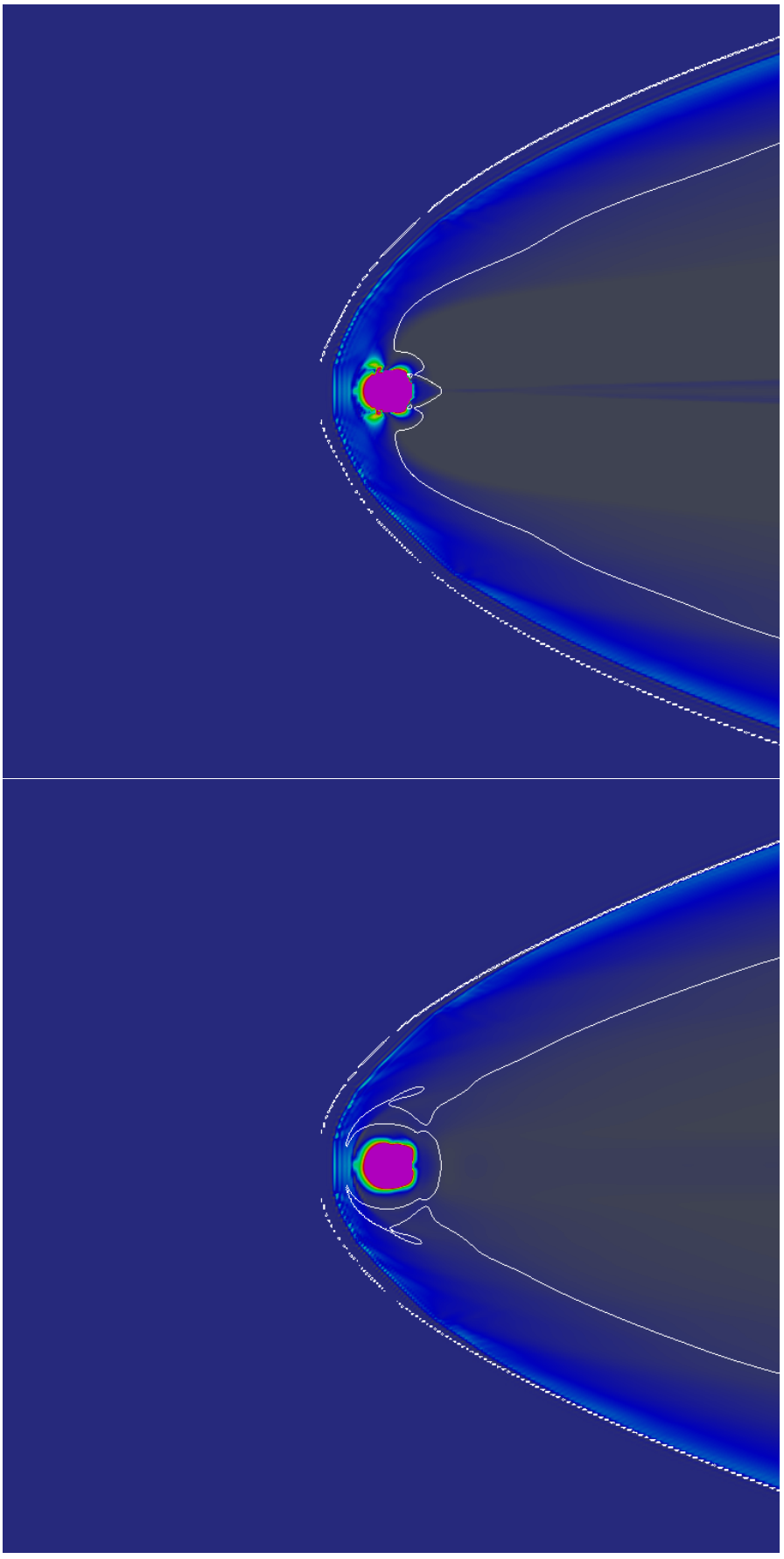

FIGURE 5.18: Comparison of the simulation results (density : ρ) between slow and fast solar wind. Results of the fast solar wind are shown where the corresponding bow shock and the magnetopause of the slow solar wind results are shown as contours. Top : X-Z plane. Bottom : X-Y plane.

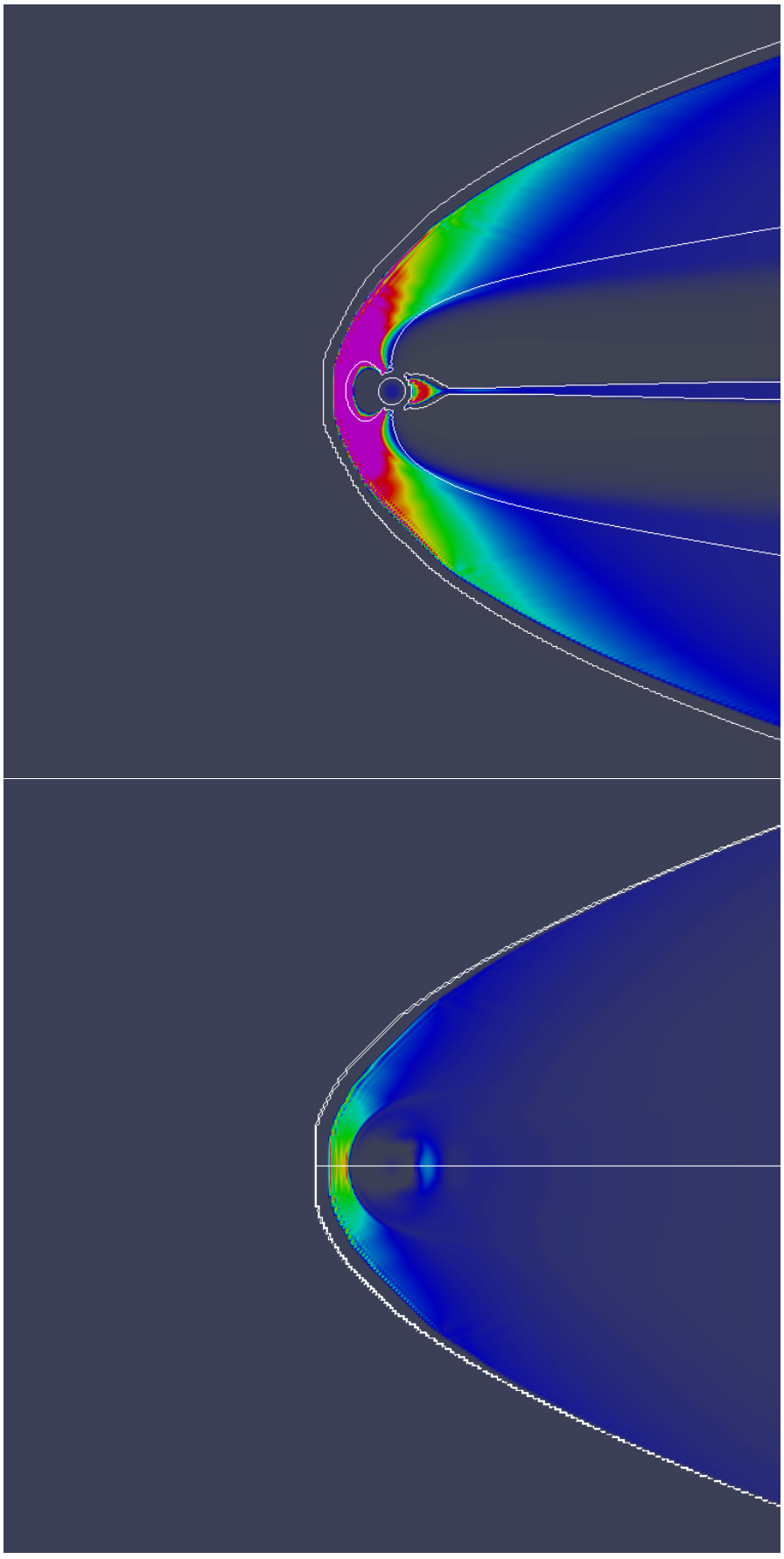

FIGURE 5.19: Comparison of the simulation results (total pressure : p) between slow and fast solar wind. Results of the fast solar wind are shown where the corresponding bow shock and the magnetopause of the slow solar wind results are shown as contours. Top : X-Z plane. Bottom : X-Y plane.

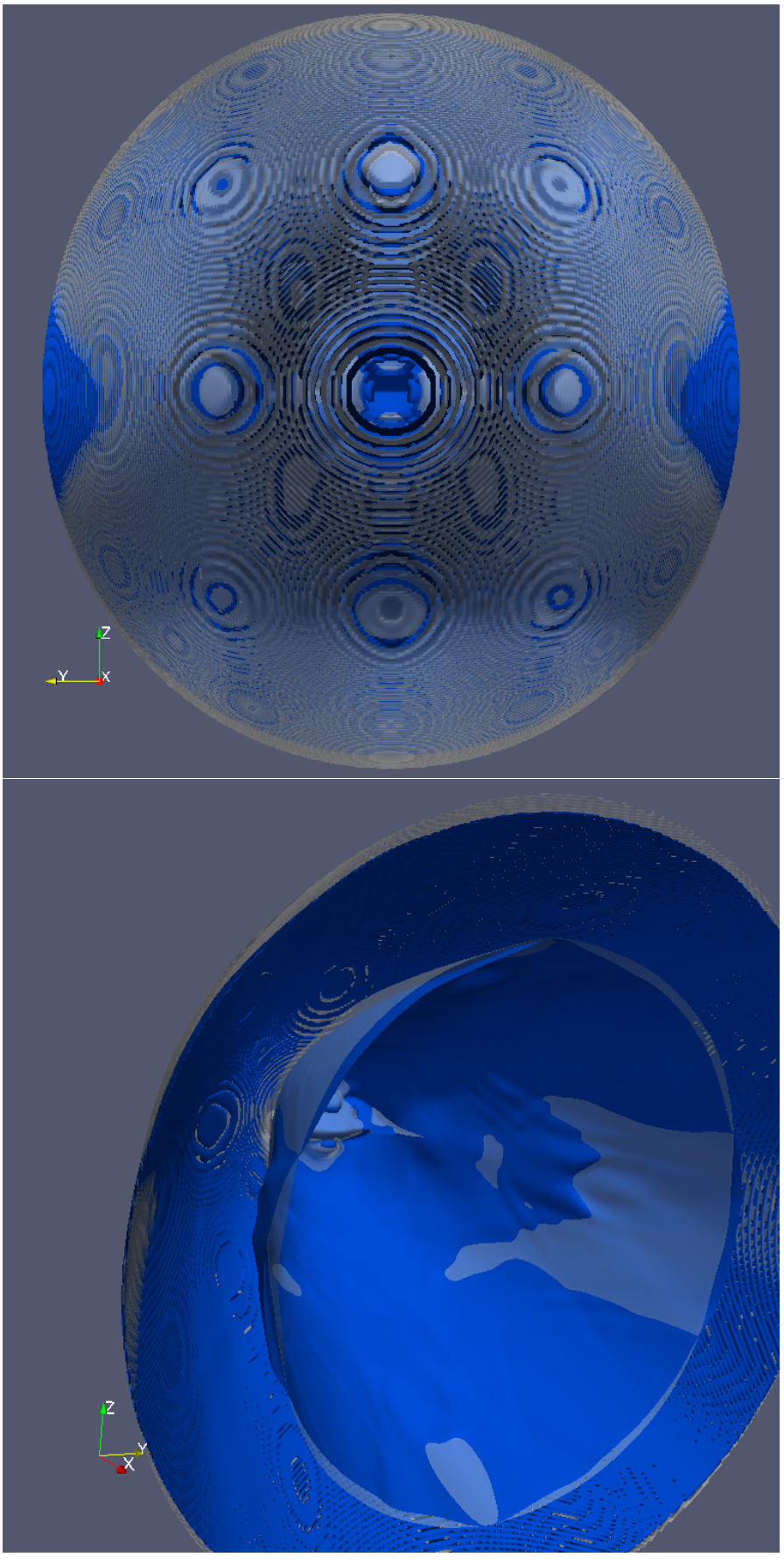

FIGURE 5.20: Comparison of the simulation results (Density ρ) between slow and fast solar wind. The isosurfaces are separated in the z-direction where they overlap in the equator plane. (Top : Front view, bottom : Back top view

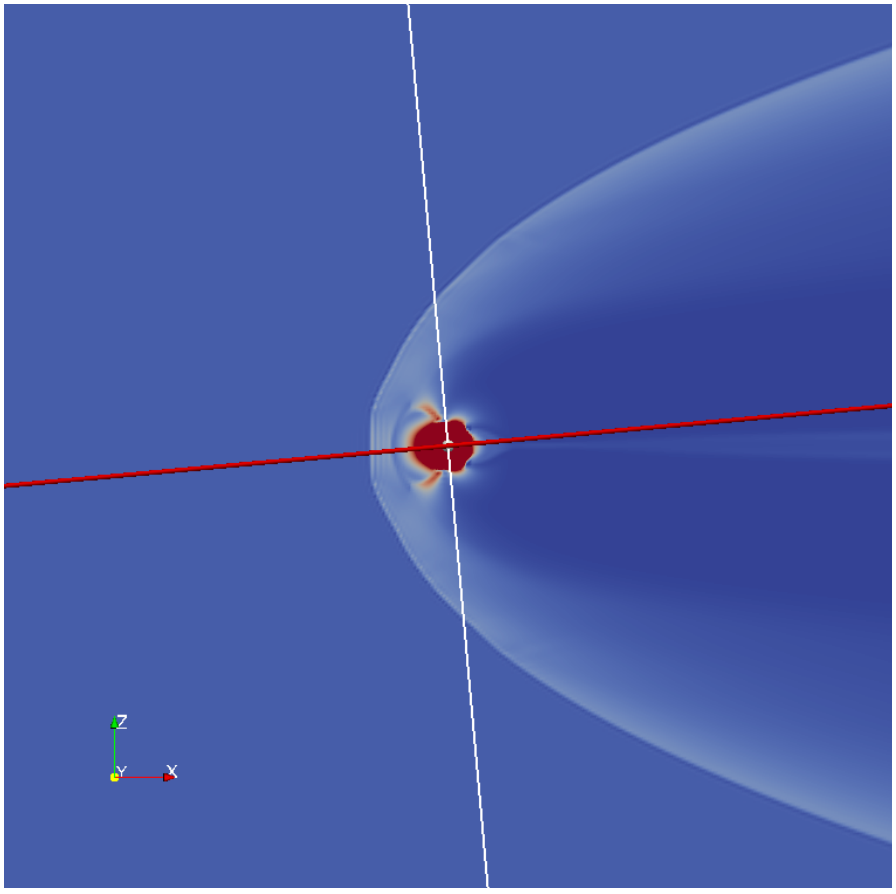

FIGURE 5.21: The red line illustrates the inclination of the Moon's orbit.

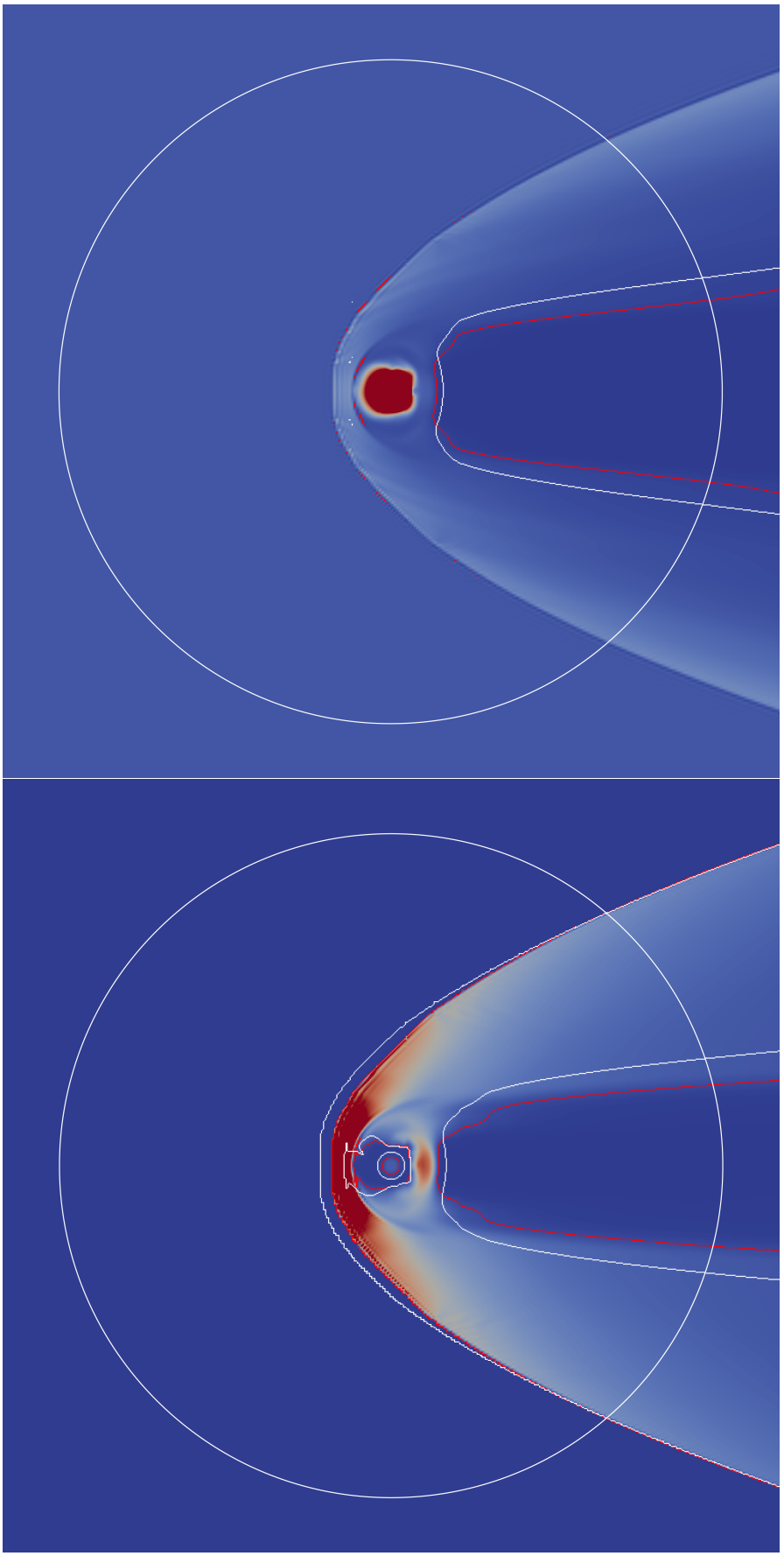

FIGURE 5.22: Comparison of the simulation results between slow and fast solar wind. The contours of the magnetopause of fast solar wind is shown in red color where it is shown in white for the slow solar wind for clear view(Top : Density ρ. Bottom : Total pressure : p. The white circle illustrates the Moon's orbit.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

After human had been successfully reach to the space, we have entered the era of space exploration. Understanding of the space environment is not only important in space exploration, but also important to space weather that interferes to our daily life and commercial community. Within the solar system, space weather is influenced by the solar wind and the interplanetary magnetic field (IMF) carried by the solar wind plasma. The Earth is protected by the magnetosphere from the charged particles of the solar wind. The Earth's magnetic field is distorted further out by the solar wind. In actuality, the solar wind is far from being a stationary flow. Observation data shows that frequent changes always occur. Space observations using detectors on spacecrafts or space telescopes are very useful but expensive, and the observed range is limited. Therefore, numerical simulations have become the powerful alternative tools in the study of the space environment. Global magnetohydrodynamic (MHD) simulation is widely used in simulating the phenomenon of the plasma environment around planets. However, the computational complexity of global MHD simulation is high. In addition, it is storage demanding. In this dissertation, we point out that the current space simulation is limited due to the computational resource.

1. The scale of space is vast. A large number of elements is required for high resolution of the mesh in global MHD simulation, which results in large memory usage.

2. The computational complexity of global MHD simulations is high. Therefore, it is time consuming to perform large-scale simulation and requires high computational power.

3. The large amount of the high resolution data is storage demanding and hard to analyze.

The above issues lead to:

1. Limitation on the size of the simulation domain.
2. Trade-off between performance and memory.
3. Simulation results can not be stored every step.

These factors decease the usability of space simulations, including global MHD simulations. We realise that not only the high computational complexity slows down the research activities when using global MHD simulation for studying the space plasma environment, but also the requirement of the memory usage limits the size of the simulation domain. As a result, specific boundary conditions have to be invoked in the simulation which affects the simulation results and the performance.

The motivation of our study is to provide an extended space simulation framework as well as its application to the investigation of the phenomenon in very large space environment. With this research we reach the following conclusions. They will be explained in detailed afterwards:

1. An efficient GPU Direct-MPI hybrid data communication framework is developed. A large-scale global MHD simulation with in-situ visualization is implemented on multi-GPU systems. High efficiency is shown in running the simulation on the GPU-rich supercomputer TSUBAME 2.5.
2. A block-based structure for efficient adaptive mesh refinement on multi-GPU systems is developed. Improvement of the global MHD simulation is made, and the restriction of the specific boundary condition is solved.
3. Simulation of solar wind interaction with the Earth's magnetosphere is conducted. Where the geomagnetic reversal is examined. Asymmetric patterns of the magnetosphere related to the inclination of the dipole field and the interplanetary magnetic field (IMF) are observed.
4. Simulation of the whole structure of the magnetosphere under slow and fast solar wind is examined. The size of the magnetosphere in the vertical direction (along north pole) and is decreased when the solar wind speed is fast. However, there is obviously variation at the equatorial plane.

5. It is found that the Moon's orbit crosses the same area beyond the bow shock under the slow and fast solar wind, but in the different areas within the magnetosheath.

6.1.1 An Efficient Large–scale MHD Simulation Code with In-situ Visualization

GPGPU (general-purpose computing on graphics processing units) and GPU computing have been widely utilized in scientific computating including MHD simulation in recent years. Our previous work — the *GPU-MHD* code [57] — achieves great speedup compared to CPU. However, the capacity of memory on a GPU board is relatively small. It limits the scale/resolution of the simulation. Scale and resolution are important issues in large-scale space simulation, especially for a phenomena occurring in wide region such as the interaction solar wind and magnetosphere . Therefore, multi-GPU systems such as the GPU-rich supercomputer TSUBAME 2.5 of the Tokyo Institute of Technology are needed to explore to perform this research. Data communication overhead between each GPU of the multi-GPU system limits the total efficiency when running large-scale simulations on GPU supercomputers. To solve this problem, an efficient GPU Direct-MPI hybrid data communication framework is presented. Large-scale global MHD simulations implemented using our GPU-Direct MPI hybrid framework is proposed. Experimental results show the significant improvement compared to a flat MPI approach. Performance tests show that our global MHD simulation achieves 4.38 TFLOPS in double precision for a computational domain of $1980 \times 1320 \times 1320$ using 216 GPUs on TSUBAME 2.5.

Visualization is indispensable in analyzing the simulation results. However, storing data for each time step of the simulation is time consuming. Writing files at every step will cause big overhead and delay the simulation process. It will also demand a lot of storage. Recording data every few steps is a feasible way but some important snapshots that may contain interesting phenomena of the simulation will be skipped. Therefore, in-situ visualization during the simulation process will be significantly helpful. In-situ visualization of the simulation results using distributed multi-GPU systems is also developed and presented. Simulation and visualization are processed using multiple GPUs simultaneously, minimizing the overheads of copying and gathering the data, only the visualization results of each partition needed to be composited for rendering. Experimental results of our global MHD simulation and visualization of solar wind interacting with the Earth's magnetosphere running 81 GPUs achieves up to 8.2 FPS and 5.0 FPS for the computational domain of $540 \times 360 \times 360$ and $810 \times 540 \times 540$ domain, respectively.

6.1.2 Advanced Global MHD Simulation using the Block-Based Structure for Efficient AMR on Multi-GPU Systems

Memory usage is an important issue that determine the size and resolution of the simulation domain. We found that the global MHD simulation for solar wind interaction with the Earth's magnetosphere has a minimum requirement of $dx \leq \frac{R_e}{2}$ at the region where the bow shock occurs. Moreover, it is found that the specific Neumann boundary condition inclined at 45 degrees to the x-axis is necessary due to the reason that not the whole bow shock (the whole magnetosphere) is included in the computational domain. The y and z boundary are on the inside the magnetosphere. Therefore, the 45 degrees shear boundary condition is necessary to avoid numerical errors. A block-based structure for efficient AMR on multi-GPU systems as well as task parallelism using multi-streaming of Compute Unified Device Architecture (CUDA) is developed. Peer-to-peer direct access is used so that no temporary buffer for the data exchange between different level grids is needed. Memory usage of the simulation is reduced while keeping a good performance of computation using GPU. Special techniques of refining the blocks only when the Lax-Wendroff step which to skip the interpolation of the values of the half grid points $((i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2}))$. We also use texture memory to store the dipole field $\mathbf{B}_d$. As a result, we are able to enlarge the simulation domain enough to simulate the full structure of the magnetosphere. Thus, we find that it is possible to replace the 45 degree shear Neumann boundary with a simple Neumann boundary (align to the y-axis and z-axis, respectively).

6.1.3 Large-scale Solar Wind Interaction with the Earht's Magnetosphere

Simulation of the solar wind interaction with the Earth's magnetosphere is presented. Features such as the bow shock, the polar cusp, magnetosheath, magnetopause and the plasma sheet are examined. The symmetric structure of the magnetosphere shows good agreement with the assumptions of [7, 23, 24]. In addition, simulations for the geomagnetic reversal is carried out by rotating the dipole field with a unit quaternion. Asymmetric results of the inclined dipole and the horizontal dipole are found. By comparing the results to the symmetric magnetosphere of the vertical dipole field, it is found that the interplanetary magnetic field (IMF) has less influence when it is parallel to the dipole due the strong magnetic field of the Earth. The different size and pressure of the polar cusp between different inclinations of the dipole field are found. A "wave-like" pattern is also found in the evolution of the magnetotail of the horizontal dipole field.

By using the advanced global MHD simulation implemented with our block-based structure and AMR, we extended the simulation domain to $(x, y, z) = (-70R_e, -70R_e, -70R_e) \sim (70R_e, 70R_e, 70R_e)$ which not only contains the whole magnetosphere but also includes the Moon's orbit. Simulations and comparisons of the Earth's magnetosphere interacting with the slow and fast solar wind are performed. It is found that the size of the Earth's magnetosphere along the pole (z-direction) is decreased if the solar wind speed is fast. On the other hand, the Earth's magnetosphere in the area of the Moon's orbit beyond the bow shock is almost the same regardless of the solar wind speed. We are looking forward to increase the simulation resolution to get more accurate results as well as the comparison to the theoretical calculation to find that difference precisely.

6.2 Future Work

Several techniques to enhance the efficiency and usability of the global MHD simulation for solar wind interaction with magnetospheres are developed and presented. Peer-to-peer data transfer and access of GPU-Direct 2.0 has contributed to the enhancement of our simulation on multi-GPU systems. Our simulations show high efficiency on the GPU-rich supercomputer TSUBAME 2.5. However, according to the specification of the computation node of TSUBAME, only two of the three GPUs can use GPU-Direct. Inter-node data communication via MPI is another 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.

The global MHD simulation as well as the modified leapfrog scheme are considered as one of the most simple simulation model applyed to the solar wind interaction with magnetosphere, where the computational complexity as well as the memory usage are relatively low. We are looking forward to implementing higher order methods such as a hybrid simulation for efficient large-scale simulation. Due to the limitation of time, only the magnetosphere of the Earth is examined, therefore, we are also looking forward to applying our simulation to other planets.

The different heights of the magnetosphere under the slow and fast solar wind is observed. The different area crossing with the Moon's orbit are also found. However, a depth study is needed to make the quantities more precisely. One remarkable result is that, by enlarging the calculation domain, we can have the whole Moon's orbit within our domain. To the author's knowledge, almost all the existing simulations of solar wind interaction only contain one astronomical body. Simulations of solar wind interaction

with the moon only has the moon placed in the center of the domain. The profile of the situation of the Moon's orbit at different location (for example, within/outside of the magnetosphere) as well as the incoming solar wind is set artificially. Therefore, we are looking forward to extending the simulation to simulate the Earth's magnetosphere with the Moon's orbit to examine the coupling between the moon and the Earth's magnetoshpere, due to the difference of the scale between the Moon and the Earth. In addition, the solar wind interacts with the lunar surface directly since the moon does not have a dipole field as the Earth does. Very high resolution (or very high level block, dynamic range) is need in the surrounding area of the moon. Therefore, improvement of our AMR is also needed to reach the target.

Appendix A

Numerical Scheme of the Ideal MHD Equations

In the TVD scheme proposed by Pen *et al.* [64, 90] for solving the ideal MHD equations, the magnetic field is held fixed first and then the fluid variables are updated. A reverse procedure is then performed to complete a one time step. Three dimensional problem is split into one-dimensional sub-problems by using a Strang-type directional splitting [65].

Firstly, we describe the fluid update step in which the fluid variables are updated while holding the magnetic field fixed. The magnetic field is interpolated to cell centers for second-order accuracy. By considering the advection along the x direction, the ideal MHD equations can be written in flux-conservative vector form as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \frac{\partial \boldsymbol{F}(\boldsymbol{u})}{\partial x} = 0 \tag{A.1}$$

Equation (A.1) is then solved by Jin & Xin's relaxing TVD method [64, 66, 90, 91]. With this method, a relaxation system — a linear system with a stiff low order term of is introduced

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + \frac{\partial}{\partial x}\mathbf{v} &= 0 \\ \frac{\partial \mathbf{v}}{\partial t} + A\frac{\partial}{\partial x}\boldsymbol{u} &= -\frac{1}{\epsilon}(\mathbf{v} - \boldsymbol{F}(\boldsymbol{u})) \end{aligned} \tag{A.2}$$

where the small positive parameter ϵ is the *relaxation rate* and

$$A = diag\{a_1, a_2, \cdots, a_n\}, \quad a_i > 0 \quad (1 \le i \le n) \tag{A.3}$$

If ϵ is sufficiently small, solving A.2 can obtain good approximation to the original conservation laws A.1. In Pen *et al.*'s approach [91], Equation A.1 is replaced by the following linear advection equation with a nonlinear stiff source term

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + \frac{\partial}{\partial x}(c\mathbf{v}) &= 0 \\ \frac{\partial \mathbf{v}}{\partial t} + \frac{\partial}{\partial x}(c\boldsymbol{u}) &= -\frac{1}{\epsilon}(\mathbf{v} - \boldsymbol{F}(\boldsymbol{u})) \end{aligned} \tag{A.4}$$

where $c(x,t)$ is called the *freezing speed*. Jin & Xin [66] considered the freezing speed to be a positive constant where Pen *et al.* [64, 90, 91] generalized it to be a free positive function. Jin & Xin [66] showed this algorithm to be TVD under the constraint that c be greater than the characteristic speed $\partial \boldsymbol{F}(\boldsymbol{u})/\partial \boldsymbol{u}$. One can now take the limit as $\epsilon \to 0$, which results in a relaxed algorithm. The linear part of Equation A.4 is decoupled through a change of variables $\mathbf{w}_1 = \boldsymbol{u} + \mathbf{v}$ and $\mathbf{w}_2 = \boldsymbol{u} - \mathbf{v}$ as the following linear equation

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial}{\partial x}(c\mathbf{w}) = 0 \tag{A.5}$$

It is solved by the monotonic upstream-centered scheme (MUSCL) and successfully implemented for simulating cosmological astrophysical fluids by Pen [91].

The MUSCL scheme for solving the linear advection equation is explicitly asymmetric since it depends on the sign of the advection velocity. A symmetrical approach that applies to a general advection velocity was proposed in [90]. The flow can be considered as a summation of a right-moving wave u^R and a left-moving wave u^L. For a positive advection velocity, the amplitude of the left-moving wave is zero. For a negative advection velocity, the amplitude of the right-moving wave is zero. In a compact representation, the right- and left-moving waves can be defined as $u^R = u(1 + v/c)/2$ and $u^L = u(1 - v/c)/2$ where $c = |v|$. The two waves are traveling in opposite directions with advection speed c and can be described by the advection equations:

$$\begin{aligned} \frac{\partial u^R}{\partial t} + \frac{\partial}{\partial x}(cu^R) &= 0 \\ \frac{\partial u^L}{\partial t} - \frac{\partial}{\partial x}(cu^L) &= 0 \end{aligned} \tag{A.6}$$

The MUSCL scheme is straightforward to apply to solve the above equations A.6 since the upwind direction is left for the right-moving wave, and it is right for the left-moving wave. The one-dimensional relaxing advection equation then becomes

$$\frac{\partial u}{\partial t} + \frac{\partial F^R}{\partial x} - \frac{\partial F^L}{\partial x} = 0 \tag{A.7}$$

where $F^R = cu^R$ and $F^L = cu^L$. Note that the relaxing advection equation will correctly reduce to the linear advection equation for any general advection velocity. Using this symmetrical approach, a general algorithm can be written to solve the linear advection equation for an arbitrary advection velocity. For ideal MHD equations, we have $\boldsymbol{u} = (u_1, u_2, u_3, u_4, u_5) = (\rho, \rho v_x, \rho v_y, \rho v_z, E)$ and the flux vector is given by

$$\boldsymbol{F} = \begin{pmatrix} \rho v_x \\ \rho v_x^2 + P^* - B_x^2 \\ \rho v_x v_y - B_x B_y \\ \rho v_x v_z - B_x B_z \\ (E + P^*) v_x - B_x \mathbf{B} \cdot \mathbf{v} \end{pmatrix} \tag{A.8}$$

A new variable $\boldsymbol{w} = \boldsymbol{F}(\boldsymbol{u})/c$ is defined as an auxiliary vector to calculate fluxes. The vector conservation law is replaced by the following relaxation system of symmetrical method

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + \frac{\partial}{\partial x}(c\boldsymbol{w}) &= 0 \\ \frac{\partial \boldsymbol{w}}{\partial t} + \frac{\partial}{\partial x}(c\boldsymbol{u}) &= 0 \end{aligned} \tag{A.9}$$

These equations can be decoupled through a change of right- and left-moving waves $\boldsymbol{u}^R = (\boldsymbol{u} + \boldsymbol{w})/2$ and $\boldsymbol{u}^L = (\boldsymbol{u} - \boldsymbol{w})/2$

$$\begin{aligned} \frac{\partial \boldsymbol{u}^R}{\partial t} + \frac{\partial}{\partial x}(c\boldsymbol{u}^R) &= 0 \\ \frac{\partial \boldsymbol{u}^L}{\partial t} - \frac{\partial}{\partial x}(c\boldsymbol{u}^L) &= 0 \end{aligned} \tag{A.10}$$

The above pair of equations is then solved by an upwind scheme, separately for right- and left-moving waves, using cell-centered fluxes. Second-order spatial accuracy is achieved by interpolating of fluxes onto cell boundaries using a monotone upwind schemes for conservation laws (MUSCL) [92] with the help of a flux limiter. Runge-Kutta scheme is used to achieve second-order accuracy of time integration.

We denote $\boldsymbol{u}_n^t$ as the cell-centered values of the cell n at time t, $\boldsymbol{F}_n^t$ as the cell-centered flux in cell n, As an example, we consider the positive advection velocity, negative direction can be obtained in a similar way. We obtain the first-order upwind flux $F_{n+1/2}^{(1),t}$ from the averaged flux F_n^t in cell n. Two second-order flux corrections can be defined

using three local cell-centered fluxes as follows

$$\begin{aligned} \Delta \boldsymbol{F}^{L,t}_{n+1/2} &= \frac{\boldsymbol{F}^t_n - \boldsymbol{F}^t_{n-1}}{2} \\ \Delta \boldsymbol{F}^{R,t}_{n+1/2} &= \frac{\boldsymbol{F}^t_{n+1} - \boldsymbol{F}^t_n}{2} \end{aligned} \tag{A.11}$$

When the corrections have opposite signs,there is no second-order correction in the case of near extrema. With the aid of a flux limiter ϕ we then get the second-order correction

$$\Delta \boldsymbol{F}^t_{n+1/2} = \phi(\Delta \boldsymbol{F}^{L,t}_{n+1/2}, \Delta \boldsymbol{F}^{R,t}_{n+1/2}) \tag{A.12}$$

The van Leer limiter [93]

$$\text{vanleer}(a,b) = \frac{2ab}{a+b} \tag{A.13}$$

is used in *GPU-MHD*. By adding the second-order correction to the first-order fluxes we obtain second-order fluxes. For example, the second-order accurate right-moving flux $\boldsymbol{F}^{R,t}_{n+1/2}$ can be calculated

$$\boldsymbol{F}^{R,t}_{n+1/2} = \boldsymbol{F}^t_n + \Delta \boldsymbol{F}^t_{n+1/2} \tag{A.14}$$

The time integration is performed by calculating the fluxes $\boldsymbol{F}(u^t_n)$ and the freezing speed c^t_n in the first half time step is given as follows

$$\boldsymbol{u}^{t+\Delta t/2}_n = \boldsymbol{u}^t_n - \left(\frac{\boldsymbol{F}^t_{n+1/2} - \boldsymbol{F}^t_{n-1/2}}{\Delta x} \right) \frac{\Delta t}{2} \tag{A.15}$$

where$\boldsymbol{F}^t_{n+1/2} = \boldsymbol{F}^{R,t}_{n+1/2} - \boldsymbol{F}^{L,t}_{n+1/2}$ is computed by the first-order upwind scheme. By using the second-order TVD scheme on $\boldsymbol{u}^{t+\Delta t/2}_n$, we obtain the full time step $\boldsymbol{u}^{t+\Delta t}_n$

$$\boldsymbol{u}^{t+\Delta t}_n = \boldsymbol{u}^t_n - \left(\frac{\boldsymbol{F}^{t+\Delta t/2}_{n+1/2} - \boldsymbol{F}^{t+\Delta t/2}_{n-1/2}}{\Delta x} \right) \Delta t \ , \tag{A.16}$$

To keep the TVD condition, the flux freezing speed c is the maximum speed information can travel and should be set to $|v_x| + (\gamma p/\rho + B^2/\rho)^{1/2}$ as the maximum speed of the fast MHD wave over all directions is chosen. As the time integration is implemented using a second-order Runge-Kutta scheme, the time step is determined by satisfying the CFL condition

$$\begin{aligned} c_{max} &= [max(|v_x|, |v_y|, |v_z|) + (\gamma p/\rho + B^2/\rho)^{1/2}] \\ \Delta t &= cfl/c_{max} \end{aligned} \tag{A.17}$$

where cfl is the Courant-Number and $cfl \lesssim 1$ is generally set to $cfl \simeq 0.7$ for stability, and B is the magnitude of the magnetic field. Constrained transport (CT) [11] is used

to keep the $\nabla \cdot \mathbf{B} = 0$ to machine precision. Therefore, the magnetic field is defined on cell faces and it is represented in arrays [64]

$$\begin{aligned} Bx(i,j,k) &= (B_x)_{i-1/2,j,k} \\ By(i,j,k) &= (B_y)_{i,j-1/2,k} \\ Bz(i,j,k) &= (B_z)_{i,j,k-1/2} \end{aligned} \tag{A.18}$$

where the cell centers are denoted by $(i,j,k) \equiv (x_i, y_j, z_k)$, and faces by $(i \pm 1/2, j, k)$, $(i, j \pm 1/2, k)$, and $(i, j, k \pm 1/2)$, etc. The cells have unit width for convenience.

Secondly, we describe the update of the magnetic field in separate two-dimensional advection-constraint steps along x-direction while holding the fluid variables fixed. The magnetic field updates along y and z-directions can be handled in a similar matter. We follow the expressions used in [94]. For example, we can calculate the averaging of v along x direction as follows

$$(v_x)_{i,j+1/2,k} = \frac{1}{4}\left[(v_x)_{i+1,j+1/2,k} + 2\,(v_x)_{i,j+1/2,k} + (v_x)_{i-1,j+1/2,k}\right] \tag{A.19}$$

A first-order accurate flux is then obtained by

$$(v_x B_y)_{i+1/2,j+1/2,k} = \begin{cases} (v_x B_y)_{i,j+1/2,k} & , \quad (v_x)_{i+1/2,j+1/2,k} > 0 \\ \\ (v_x B_y)_{i+1,j+1/2,k} & , \quad (v_x)_{i+1/2,j+1/2,k} \leq 0 \end{cases} \tag{A.20}$$

where the velocity average is

$$(v_x)_{i+1/2,j+1/2,k} = \frac{1}{2}\left[(v_x)_{i,j+1/2,k} + (v_x)_{i+1,j+1/2,k}\right] \tag{A.21}$$

B_x is updated by constructing a second-order-accurate upwind electromotive force (EMF) $v_y B_x$ using Jin & Xin's relaxing TVD method [66] in the advection step. Then this same EMF is immediately used to update B_y in the constraint step.

Extension to three dimensions can be done through a Strang-type directional splitting [65]. Equation (A.1) is dimensionally split into three separate one-dimensional equations. For a time step Δt, let $fluid_x$ be the fluid update along x, $B_{x\to y}$ be the update of B_x along y, and L_i be the update operator of $\boldsymbol{u}^t$ to $\boldsymbol{u}^{t+\Delta t}$ by including the flux along i direction. Each L_i includes three update operations in sequence, for example, L_x includes $fluid_x$, $B_{y\to x}$, and $B_{z\to x}$. A forward sweep and a reverse sweep are defined as $\boldsymbol{u}^{t+\Delta t} = L_z L_y L_x \boldsymbol{u}^t$ and $\boldsymbol{u}^{t+2\Delta t} = L_x L_y L_z \boldsymbol{u}^{t+\Delta t}$, respectively. A complete update combines a forward sweep and reverse sweep. The dimensional splitting of the relaxing

TVD can be expressed as follows [90]

$$\boldsymbol{u}^{t_2} = \boldsymbol{u}^{t_1+2\Delta t_1} = L_x L_y L_z L_z L_y L_x \boldsymbol{u}^{t_1} \tag{A.22}$$

$$\boldsymbol{u}^{t_3} = \boldsymbol{u}^{t_2+2\Delta t_2} = L_z L_x L_y L_y L_x L_z \boldsymbol{u}^{t_2} \tag{A.23}$$

$$\boldsymbol{u}^{t_4} = \boldsymbol{u}^{t_3+2\Delta t_3} = L_y L_z L_x L_x L_z L_y \boldsymbol{u}^{t_3} \tag{A.24}$$

where Δt_1, Δt_2, and Δt_3 are sequential time steps after each double sweep.

Appendix B

The Modified Leapfrog Scheme

The modified leapfrog scheme is a finite difference method (FDM) which is a combination of the Lax-Wendroff scheme and leapfrog scheme. We provide the details of the modified leapfrog scheme in the following section. For a simple partial differential equation,

$$\frac{\partial f}{\partial t} = -\frac{\partial f}{\partial x} - \frac{\partial f}{\partial y} - \frac{\partial f}{\partial z} - f \tag{B.1}$$

the two-step Lax-Wendroff scheme is listed as follows. The 1st step is obtained by

$$\begin{aligned} f^t_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} =& \frac{1}{8}(f^t_{i,j,k} + f^t_{i+1,j,k} + f^t_{i,j+1,k} + f^t_{i,j,k+1} \\ &+ f^t_{i+1,j+1,k} + f^t_{i+1,j,k+1} + f^t_{i,j+1,k+1} + f^t_{i+1,j+1,k+1}) \end{aligned} \tag{B.2}$$

$$\begin{aligned} f^{t+\frac{1}{2}}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} =& f^t_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} - \frac{\Delta t}{2} f^t_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} \\ &- \frac{\Delta t}{8\Delta x}(f^t_{i+1,j+1,k+1} + f^t_{i+1,j+1,k} + f^t_{i+1,j,k+1} + f^t_{i+1,j,k} \\ &- f^t_{i,j,k} - f^t_{i,j+1,k} - f^t_{i,j,k+1} - f^t_{i,j+1,k+1}) \\ &- \frac{\Delta t}{8\Delta y}(f^t_{i+1,j+1,k+1} + f^t_{i+1,j+1,k} + f^t_{i,j+1,k+1} + f^t_{i,j+1,k} \\ &- f^t_{i,j,k} - f^t_{i+1,j,k} - f^t_{i,j,k+1} - f^t_{i+1,j,k+1}) \\ &- \frac{\Delta t}{8\Delta z}(f^t_{i+1,j+1,k+1} + f^t_{i+1,j,k+1} + f^t_{i,j+1,k+1} + f^t_{i,j,k+1} \\ &- f^t_{i,j,k} - f^t_{i+1,j,k} - f^t_{i,j+1,k} - f^t_{i+1,j+1,k}) \end{aligned} \tag{B.3}$$

The 2nd step is obtained by

$$\begin{aligned} f^{t+\frac{1}{2}}_{i,j,k} =& \frac{1}{8}(f^{t+\frac{1}{2}}_{i-\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}} + f^{t+\frac{1}{2}}_{i+\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}} + f^{t+\frac{1}{2}}_{i-\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}} + f^{t+\frac{1}{2}}_{i-\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}} \\ &+ f^{t+\frac{1}{2}}_{i+\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}} + f^{t+\frac{1}{2}}_{i+\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}} + f^{t+\frac{1}{2}}_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} + f^{t+\frac{1}{2}}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} \end{aligned} \tag{B.4}$$

$$
\begin{aligned}
f_{i,j,k}^{t+1} =& f_{i,j,k}^{t} - \Delta t f_{i,j,k}^{t+\frac{1}{2}} \\
& - \frac{\Delta t}{4\Delta x}(f_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}} + f_{i+\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}} + f_{i+\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}} + f_{i+\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}} \\
& - f_{i-\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}} - f_{i-\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}} - f_{i-\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}} - f_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}}) \\
& - \frac{\Delta t}{4\Delta y}(f_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}} + f_{i+\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}} + f_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}} + f_{i-\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}} \\
& - f_{i-\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}} - f_{i+\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}} - f_{i-\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}} - f_{i+\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}}) \\
& - \frac{\Delta t}{4\Delta z}(f_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}} + f_{i+\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}} + f_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}} + f_{i-\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}}^{t+\frac{1}{2}} \\
& - f_{i-\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}} - f_{i+\frac{1}{2},j-\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}} - f_{i-\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}} - f_{i+\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}}^{t+\frac{1}{2}})
\end{aligned}
\tag{B.5}
$$

The only difference between the two-step Lax-Wendroff scheme and the leapfrog scheme is that the leapfrog scheme uses $\Delta t f_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}^{t-\frac{1}{2}}$ instead of $\frac{\Delta t}{2} f_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}^{t}$ in Equation (B.3). The leapfrog scheme uses the result of last time step $(t - \frac{\Delta t}{2})$ where the two-step Lax-Wendroff uses the interpolation between the n and $n+1$ grid-points in the same time step t. Calculations of a time step of these two schemes are shown in Figure B.1. Figure B.2 shows the whole procedure of the modified leapfrog scheme. For each sequence l, the two-step Lax-Wendroff scheme is used for the 1^{st} time step and the leapfrog scheme is used for the subsequent $l - 1$ time steps.

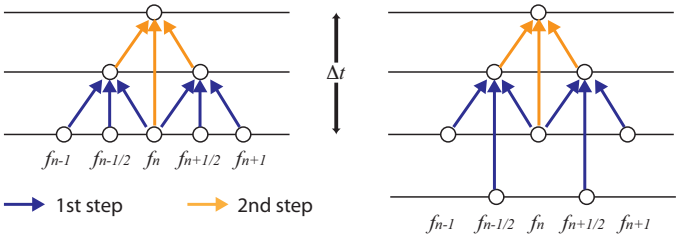

FIGURE B.1: Calculations of one time step of the two-step Lax-Wendroff scheme (left) and the leapfrog scheme (right).

Experimental results show that the modified modified leapfrog scheme is better then the two-step Lax-Wendroff scheme and the leapfrog scheme in controlling the numerical oscillation. Comparison results of the Brio-Wu shock tube test between the three schemes are shown in Figure B.3.

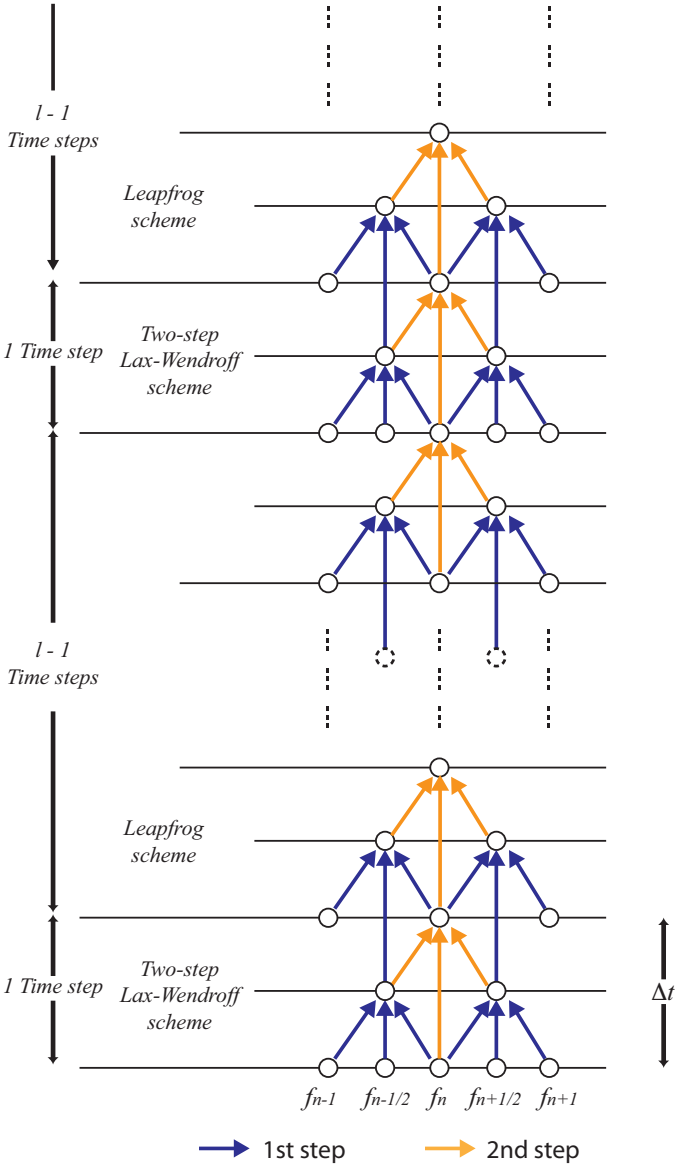

FIGURE B.2: Illustration of the modified leapfrog scheme.

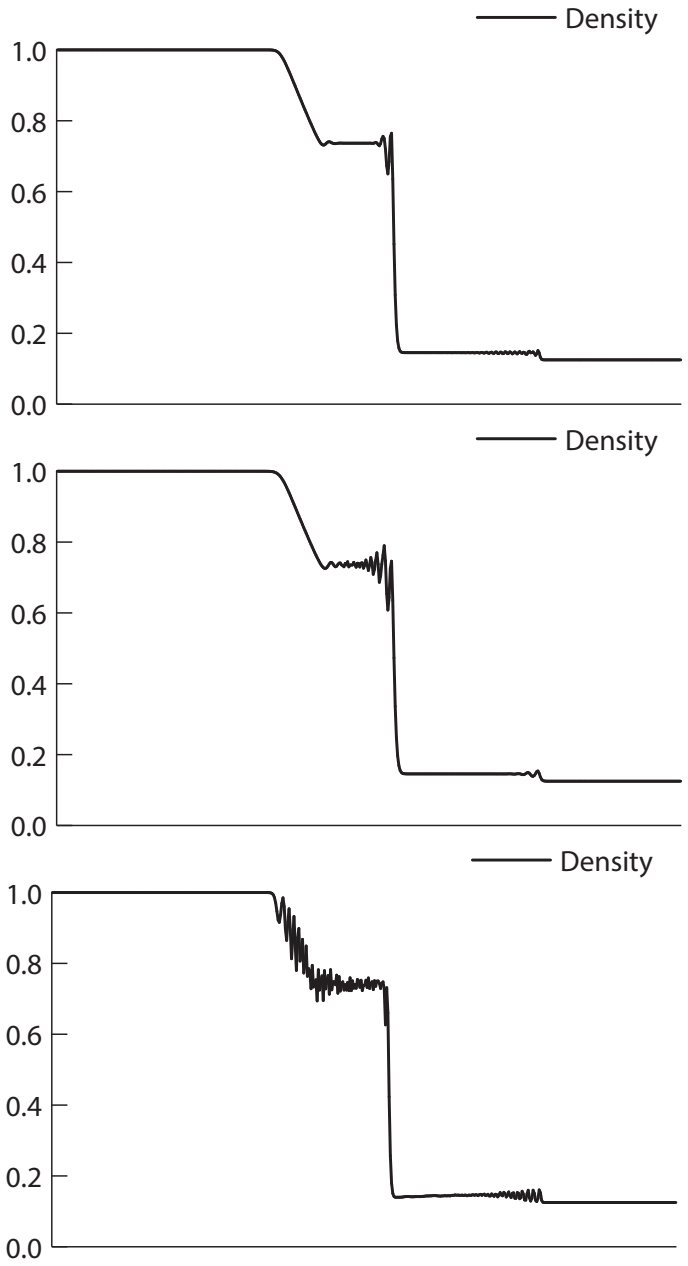

FIGURE B.3: Comparison between the modified leapfrog scheme (top), the two-step Lax-Wendroff scheme (middle) and the leapfrog scheme (bottom).

Bibliography

[1] Space weather prediection center, national oceanic and atmospheric administration. URL http://www.swpc.noaa.gov/products/ace-real-time-solar-wind.

[2] Cuda zone. URL https://developer.nvidia.com/category/zone/cuda-zone.

[3] Global scientific information and computing center, Tokyo Institute of Technology. TSUBAME 2.5 hardware and software specification. URL http://www.gsic.titech.ac.jp/sites/default/files/spec25e1.pdf.

[4] G. Tóth. The $\nabla \cdot \mathbf{b} = 0$ constraint in shock-capturing magnetohydrodynamics codes. *Journal of Computational Physics*, 161:605–656, 2000.

[5] Spherical blast wave test page of Athena3D test suite:. URL http://www.astro.virginia.edu/VITA/ATHENA/blast.html.

[6] D. Lee and A. E. Deane. An unsplit staggered mesh scheme for multidimensional magnetohydrodynamics. *Journal of Computational Physics*, 228:952–975, 2009.

[7] T. Ogino, R. J. Walker, and M. Ashour-Abdalla. A global magnetohydrodynamic simulation of the response of the magnetosphere to a northward turning of the interplanetary magnetic field. *Journal of Geophysical Research*, 99:11,027–11,042, 1994.

[8] The National Space Weather Program. URL http://www.nswp.gov.

[9] T. Umeda and K. Fukazawa. A high-resolution global Vlasov simulation of a small dielectric body with a weak intrinsic magnetic field on the k computer. *Earth, Planets and Space*, 67(1):49(8 pages), 2015.

[10] J. P. Goedbloed and S. Poedts. *Principles of Magnetohydrodynamics*. Cambridge University Press, 2004.

[11] J. F. Hawley C. R. Evans. Simulation of magnetohydrodynamic flow: a constrained transport method. *The Astrophysical Journal*, 332:659–677, Sept 1988.

[12] A. R. Barakat and R. W. Schunk. Transport equations for multicomponent anisotropic space plasmas: A review. *Plasma PhysicsEmail alert RSS feed*, 24 (4):389, 1982.

[13] J. Stone and T. A. Gardiner. Recent progress in astrophysical MHD. *Computer Physics Communications*, 177:52–59, 2007.

[14] J. Stone, T. A. Gardiner, P. Teuben, J. F. Hawley, and J. B. Simon. Athena: a new code for astrophysical MHD. *Astrophysical Journal Supplement Series*, 178: 137–177, 2008.

[15] Xueshang Feng, Yanqi Hu, and Fengsi Wei. Modeling the resistive mhd by the cese method. *Solar Physics*, 235:235–257, 2006.

[16] X. Feng, Y. Zhou, and S. T. Wu. A novel numerical implementation for solar wind modeling by the modified conservation element/solution element method. *Astrophysical Journal*, 655:1110–1126, 2007.

[17] M. Dryer. Multidimensional, magnetohydrodynamic simulation of solar-generated disturbances: Space weather forecasting of geomagnetic storms. *AIAA Journal*, 36 (3):365–370, 1998.

[18] M. Dryer. Space weather simulations in 3D MHD from the sun to earth and beyond to 100 au: a modeler's perspective of the present state of the art (invited review). *Asian J. of Physics*, 16:97–121, 2007.

[19] J. N. Leboeuf, T. Tajima, C. F. Kennel, and J. M. Dawson. Global simulation of the time-dependent magnetosphere. *Geophysical Research Letters*, 5:609–612, 1978.

[20] J. N. Leboeuf, T. Tajima, C. F. Kennel, and J. M. Dawson. Global simulations of the three-dimensional magnetosphere. *Geophysical Research Letters*, 8:257–260, 1981.

[21] J. Raeder. *Space Plasma Simulation*, chapter Global magnetohydrodynamics — A tutorial. Springer, 2003.

[22] M. Palmroth, P. Janhunen, T. I. Pulkkinen, and W. K. Peterson. Cusp and magnetopause locations in global MHD simulation. *Journal of Geophysical Research: Space Physics*, 106:29435–29450, 2001.

[23] T. Ogino. A three-dimensional mhd simulation of the interaction of the solar wind with the earth's magnetosphere: The generation of field-aligned currents. *Journal of Geophysical Research*, 91:6791–6806, 1986.

[24] 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 Transactions on Plasma Science*, 20, 1992.

[25] T. Ogino, R. J. Walker, and M. G. Kivelson. A global magnetodynamic simulation of the Jovian magnetosphere. *Journal of Geophysical Research*, 103:225–235, 1998.

[26] K. Fukazawa, T. Ogino, and R. J. Walker. Dynamics of the Jovian magnetosphere for northward interplanetary magnetic field (IMF). *Geophysical Research Letters*, 32:L03202, 2005.

[27] K. Fukazawa, T. Ogino, and R. J. Walker. Configuration and dynamics of the Jovian magnetosphere. *Journal of Geophysical Research*, 111:A10207, 2006.

[28] T. Ogino K. Fukazawa and R. J. Walker. A simulation study of dynamics in the distant Jovian magnetotail. *Journal of Geophysical Research*, 115:A09219, 2010.

[29] K. Fukazawa, S. Ogi, T. Ogion, and R. J. Walker. Magnetospheric convection at Saturn as a function of IMF B_Z. *Geophysical Research Letters*, 34:L01105, 2007.

[30] K. Fukazawa, T. Ogino, and R. J. Walker. Vortex-associated reconnection for northward IMF in the Kronian magnetosphere. *Geophysical Research Letters*, 34: L23201, 2007.

[31] R. J. Walker, K. Fukazawa, T. Ogino, and D. Morozoff. A simulation study of Kelvin-Helmholtz waves at saturn's magnetopause. *Journal of Geophysical Research*, 116:A03203, 2011.

[32] K. Fukazawa, T. Ogino, and R. J. Walker. A magnetohydrodynamic simulation study of Kronian field-aligned currents and auroras. *Journal of Geophysical Research*, 117:A02214, 2012.

[33] Wing-Huen Ip and Andreas Kopp. MHD simulations of the solar wind interaction with Mercury. *Journal of Geophysical Research*, 107, 2002.

[34] LiangHai XIE, Lei LI, YiTeng ZHANG, and Darren Lee De ZEEUW. Three-dimensional MHD simulation of the lunar wake. *Science China Earth Science*, 56:330–338, 2013.

[35] Hongang Yang, Shuping Jin, and Chaoxu Liu. Hall MHD reconnection with an initial guide field b_{y0}. *Advances in Space Research*, 41:Pages 1649–1657, 2008.

[36] C. X. Liu, S. P. Jin, F. S. Wei, Q. M. Lu, and H. A. Yang. Plasmoid-like structures in multiple x line hall mhd reconnection. *Journal of Geophysical Research: Space Physics*, 114, 2009.

[37] Yingjuan Ma, Andrew F. Nagy, Kenneth C. Hansen, Darren L. DeZeeuw, , Tamas I. Gombosi, and K. G. Powell. Three-dimensional multispecies mhd studies of the solar wind interaction with mars in the presence of crustal fields. *Journal of Geophysical Research*, 107, 2002.

[38] Iver H. Cairns and J. G. Lyon. MHD simulations of earth's bow shock at low mach number: Standoff distances. *Journal of Geophysical Research*, 100(A9):17173–17180, 1995.

[39] J. F. Chapman, Iver H. Cairns, J. G. Lyon, and Christopher R. Boshuizen. MHD simulations of earth's bow shock: Interplanetary magnetic field orientation effects on shape and position. *Journal of Geophysical Research*, 109, 2004.

[40] X. C. Guo, C. Wang, Y. Q. Hu, and J. R. Kan. Bow shock contributions to region 1 field-aligned current: A new result from global MHD simulations. *GEOPHYSICAL RESEARCH LETTERS*, 35, 2008.

[41] H. Burau, R. Widera, W. Hönig, G. Juckeland, T. Kluge A. Debus, U. Schramm, T. E. Cowan, R. Sauerbrey, and M. Bussmann. PIConGPU: A fully relativistic particle-in-cell code for a GPU cluster. *IEEE Transactions on Plasma Science*, 38: 2831–2839, 2010.

[42] Takashi Shimokawabe, Takayuki Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama, and S. Matsuoka. An 80-fold speedup, 15.0 TFLOPS full GPU acceleration of non-hydrostatic weather model asuca production code. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2010.

[43] Takashi Shimokawabe, Takayuki Aoki, T. Takaki, A. Yamanaka, A. Nukada, T. Endo, N. Maruyama, and S. Matsuoka. Peta-scale phase-field simulation for dendritic solidification on the TSUBAME 2.0 supercomputer. In *Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, number 3, 2011.

[44] Xian Wang and Takayuki Aoki. Multi-GPU performance of incompressible flow computation by lattice boltzmann method on gpu cluster. *Parallel Computing*, 37, 2011.

[45] S. Bastrakov, R. Donchenko, A. Gonoskov, E. Efimenko, A. Malyshev, I. Meyerov, and I. Surmin. Particle-in-cell plasma simulation on heterogeneous cluster systems. *Journal of Computational Science*, pages 474–479, 2012.

[46] A. Lani, M.S. Yalim, and S. Poedts. A GPU-enabled finite volume solver for global magnetospheric simulations on unstructured grids. *Computer Physics Communications*, 185:2538–2557, 2014.

[47] Top 500 list. URL http://www.top500.org.

[48] The Green 500 list. URL http://http://www.green500.org.

[49] A. Yamanaka, T. Aoki, S. Ogawa, and T. Takaki. GPU-accelerated phase-field simulation of dendritic solidification in a binary alloy. *Journal of Crystal Growth*, 318:40–45, 2011.

[50] D. B. Kirk and W. W. Hwu. *Programming Massively Parallel Processors*. Addison-Wesley, second edition edition, 2013.

[51] G. Stantchev, W. Dorland, and N. Gumerov. Fast parallel-to-grid interpolation for plasma PIC simulations on the GPU. *Journal of Parallel and Distributed Computing*, 68:1339–1349, 2008.

[52] G. Stantchev, D. Juba, W. Dorland, and A. Varshney. Using graphics processors for high-performance computation and visualization of plasma turbulence. *Computing in Science and Engineering*, 11:52–59, 2009.

[53] P. Abreu, R. A. Fonseca, J. M. Pereira, and L. O. Silva. PIC codes in new processors: A full relativistic PIC code in CUDA-enabled hardware with direct visualization. *IEEE Transactions on Plasma Science*, 39:675–685, 2011.

[54] X. Kong, M. C. Huang, C. Ren, and V. K. Decyk. Particle-in-cell simulations with charge-conserving current deposition on graphic processing units. *Journal of Computational Physics*, 230:1676–1685, 2011.

[55] V. K. Decyk and T. V. Singh. Adaptable particle-in-cell algorithms for graphical processing units. *Computer Physics Communications*, 182:641–648, 2011.

[56] K. Madduri, E.-J. Im, K. Z. Ibrahim, S. Williams, S. Ethier, and L. Oliker. Gyrokinetic particle-in-cell optimization on emerging multi- and manycore platforms. *Parallel Computing*, 37:501–520, 2011.

[57] Hon-Cheng Wong, Un-Hong Wong, Xueshang Feng, and Zesheng Tang. Efficient magnetohydrodynamic simulations on graphics processing units with CUDA. *Computer Physics Communications*, 182:2132–2160, 2011. doi: 10.1016/j.cpc.2011.05.011.

[58] B. Pang, U.-L. Pen, and M. Perrone. Magnetohydrodynamics on heterogeneous architectures: a performance comparison. arXiv:1004.1680, 2010.

[59] P. Wang, T. Abel, and R. Kaehler. Adaptive mesh fluid simulations on GPU. *New Astronomy*, 15:581–589, 2010.

[60] R. Ueda, Y. Matsumoto, M. Itagaki, and S.-I. Oikawa. Effectiveness of GPGPU for solving the magnetohydrodynamics equations usng the CIP-MOCCT method. *Plasma and Fusion Research: Regular Articles*, 6(Article 24010192), 2011.

[61] B. Zink. HORIZON: Accelerated general relativistic magnetohydrodynamics. arXiv:1102.5202, 2011.

[62] L. Lin, C. S. Ng, and A. Bhattacharjee. Large-scale high-lundquist number reduced MHD simulations of the solar corona using gpu accelerated machines. arXiv:1109.6038, 2011.

[63] A. Wasiljew and K. Murawski. A new CUDA-based GPU implementation of the two-dimensional Athena code. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, 61:239–250, 2013.

[64] U.-L. Pen, P. Arras, and S. Wong. A free, fast, simple and efficient TVD MHD code. *The Astrophysical Journal Supplement Series*, 149(2):447–455, 2003.

[65] G. Strang. On the construction and comparison of difference schemes. *SIAM Journal on Numerical Analysis*, 5(3):506–517, Sep 1968.

[66] Z. Xin S. Jin. The relaxation schemes for systems of conservation laws in arbitrary space dimensions. *Communications on Pure and Applied Mathematics*, 48:235–276, 1995.

[67] Graham Sellers, Richard S. Wright Jr., and Nicholas Haemel. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Addision-Wesley, 2013.

[68] K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real time volume graphics*. A K Peters, Wellesley, 2006.

[69] Nvidia's next generation CUDA compute architecture: Kepler GK110, whitepaper, 2012. URL http://www.nvidia.com/content/pdf/kepler/nvidia-kepler-gk110-architecture-whitepaper.pdf.

[70] Nvidia's next generation CUDA compute architeture: Fermi, whitepaper, 2009. URL http://www.nvidia.com/content/pdf/fermi_white_papers/nvidiafermicomputearchitecturewhitepaper.pdf.

[71] T. Endo, A. Nukada, S. Matsuoka, and N. Maruyama. LINPACK evaluation on a supercomputer with heterogeneous accelerators. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10)*, pages 1–8, Atlanta, GA, USA, 2010. IEEE.

[72] A. Zachary, A. Malagoli, and P. Colella. A higher-order godunov method for multidimensional ideal magnetohydrodynamics. *SIAM Journal of Scientific Computing*, 15:263–284, 1994.

[73] A. Orszag and C. M. Tang. Small-scale structure of two-dimensional magnetohydrodynamics turbulence. *Journal of Fluid Mechanics*, 90:129–143, 1979.

[74] P. D. Mininni, A. G. Pouquet, and D. C. Montgomery. Small-scale structures in three-dimensional magnetohydrodynamic turbulence. *Physical Review Letters*, 97 (Article 244503), 2006.

[75] Un-Hong Wong, Takayuki Aoki, and Hon-Cheng Wong. Efficient magnetohydrodynamic simulations on distributed multi-GPU systems using a novel GPU Direct-MPI hybrid approach. *Computer Physics Communications*, 185:1901–1913, 2014. doi: 10.1016/j.cpc.2014.03.018.

[76] T. Ogino. Three-dimensional global MHD simulation code for the Earth's magnetosphere using HPF/JA. *Concurrency and Computation: Practice and Experience*, 2002.

[77] T. Ogino. *Space Plasma Simulation*, chapter Global magnetohydrodynamic simulation using high performance FORTRAN on parallel computers, pages 296–314. Springer, 2003.

[78] Z. Huang, C. Wang, Y. Hu, and X. Guo. Parallel implementation of 3D global MHD simulations for Earth's magnetosphere. *Computers and Mathematics with Applications*, 55:419–425, 2008.

[79] K. Reuter, F. Jenko, C. B. Forest, and R. A. Bayliss. A parallel implementation of an MHD code for the simulation of mechanically driven, turbulent dynamos in spherical geometry. *Computer Physics Communications*, 179:245–249, 2008.

[80] U. Ziegler. The NIRVANA code: parallel computational MHD with adaptive mesh refinement. *Computer Physics Communications*, 179:227–244, 2008.

[81] K. Fukazawa, T. Umeda, T. Miyoshi, N. Terada, Y. Matsumoto, and T. Ogino. Performance measurement of magneto-hydro-dynamic code for space plasma on the various scalar type supercomputer systems. *IEEE Transactions on Plasma Science*, 38:22–54, 2010.

[82] 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 High Performance Computing Applications*, 2012.

[83] Un-Hong Wong, Hon-Cheng Wong, and Yonghui Ma. Global magnetohydrodynamic simulations on multiple gpus. *Computer Physics Communications*, 185:144–152, 2014. doi: 10.1016/j.cpc.2013.08.027.

[84] D. Potter. *Computational Physics*. John Wiley and Sons, 1973.

[85] J. B. Kuipers. *Quaternions and Rotation Sequences: A Primer With Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, 2002.

[86] Kwan-Liu Ma. In-situ visualization at extreme scale: Challenges and opportunities. *Computer Graphics and Applications*, 29:14–19, November 2009.

[87] Hongfeng Yu, Chaoli Wang, Ray W. Grout, Jacqueline H. Chen, and Kwan-Liu Ma. In-situ visualization for large-scale combustion simulations. *Computer Graphics and Applications*, 30:45–57, May 2010.

[88] Janine C. Bennett, Hasan Abbasi, Peer-Timo Bremer, Ray Grout, Attila Gyulassy, Tong Jin, Scott Klasky, Hemanth Kolla, Manish Parashar, Valerio Pascucci, Philippe Pebay, David Thompson, Hongfeng Yu, Fan Zhang, and Jacqueline Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proceedings of the 2012 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–9, 2012.

[89] Hsi-Yu Schive, Yu-Chih Tsai, and Tzihong Chiueh. GAMER: A graphic processing unit accelerated adaptive-mesh-refinement code for astrophysics. *The Astrophysical Journal Supplement Series*, 186(2):457–484, 2010.

[90] H. Trac and U.-L. Pen. A primer on Eulerian computational fluid dynamics for astrophysics. *Publications of the Astronomical Society of the Pacific*, 115:303–321, 2003.

[91] U.-L. Pen. A high-resolution adaptive moving mesh hydrodynamic algorithm. *The Astrophysical Journal Supplement Series*, 115:19–34, March 1998.

[92] B. van Leer. Towards the ultimate convervative difference scheme v. a second-order sequel to godunov's method. *Journal of Computational Physics*, 32:101–136, 7 1979.

[93] B. van Leer. Towards the ultimate conservative difference scheme ii. monotonicity and conservation combined in a second order scheme. *Journal of Computational Physics*, 14:361–370, 1974.

[94] R. Käppeli, S. C. Whitehouse, S. Scheidegger, U.-L. Pen, and M. Liebendörfer. Fish: a 3D parallel MHD code for astrophysical applications. arXiv:0910.2854v1, 2009.