Computer Physics Communications 238 (2019) 181–193

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

A multi-GPU finite volume solver for magnetohydrodynamics-based solar wind simulations

Yuan Wang [a,b], Xueshang Feng [a,*], Yufen Zhou [a], Xinbiao Gan [c]

[a] SIGMA Weather Group, State Key Laboratory for Space Weather, National Space Science Center, Chinese Academy of Sciences, Beijing 100190, China
[b] College of Earth Sciences, University of Chinese Academy of Sciences, Beijing 100049, China
[c] School of Computer, National University of Defense Technology, China

ARTICLE INFO

Article history:
Received 5 December 2017
Received in revised form 27 October 2018
Accepted 4 December 2018
Available online 19 December 2018

Keywords:
MHD
FV method
Spherical shell domain
CUDA
Solar wind

ABSTRACT

Magnetohydrodynamic (MHD) simulations in the domain of spherical shell are a crucial and challenging subject in many fields such as geophysics and solar-terrestrial physics, due to the complication of the MHD equations and the specificity of the domain. Besides, due to the real-time requirement, accelerating the heavy computation is proposed in many practical problems, of which the space weather simulation and forecast from the Sun to Earth is a typical case. Considering these factors, we first develop a new, spatially second-order accurate finite volume (FV) solver for three-dimensional (3D) MHD simulations with the multiple time steps strategy, which is based on the six-component grid for spherical shell domain. Then to speed up the simulation, we implement the solver on multiple GPUs with optimizations of CUDA and establish an effective multi-GPU FV solver on the spherical shell domain. A MHD manufactured solution is used to validate the solvers' spatial accuracy, and to measure their performances. Results show that both solvers have nice scalability, and speedup ratios of 27.7x to 30.06x are obtained on GPUs. Then we utilize them to study the ambient solar wind for Carrington rotation (CR) 2060. The multi-GPU FV solver can not only obtain speedup ratios of about 29.0x, but capture main features of the solar corona and the mapped in-situ solar wind measurements.

© 2018 Published by Elsevier B.V.

1. Introduction

Magnetohydrodynamic (MHD) simulations in the domain of spherical shell are a crucial and challenging subject in many fields such as geophysics and solar-terrestrial physics. On the one hand, it is complicated to numerically solve the MHD equations combining the equations of gas dynamics with the Maxwell equations, among which the violation of the divergence constraint $\nabla \cdot \mathbf{B} = 0$ may cause extra unphysical force parallel to the magnetic field and break down numerical computation. On the other hand, the specificity of the domain enhances this complication for its spherical shape. Various grid systems have been studied including the latitude–longitude grid [1–7] and Cartesian or cylindrical system [8–11]. Ronchi et al. [12] presented the cubed sphere method that projects the sides of a circumscribed cube onto a spherical surface and divides the sphere into six identical regions. By choosing the coordinate lines on each region to be arcs of great circles, six coordinate systems are obtained and do not overlap with each other. Feng et al. [13] introduced the six-component grid for solar wind MHD modeling, which is a composite mesh consisting of six identical components to envelope the spherical surface. Each component is just a low latitude region of the usual latitude–longitude grid, with partial overlap on the component boundaries. Both the cubic sphere grid and the six-component grid show advantages in sphere-surface body-fitting and parallelization.

In addition, another considerable challenge is about computing time and memory storage. Three-dimensional (3D) MHD simulations present a large and different temporal and spatial scales which cause heavy calculation and high arithmetic intensity, especially for space-physics problems. A typical case is that the numerical space weather modeling from the Sun to Earth or beyond is feasible only on massively parallel computers for the sake of computational resources [14]. Moreover, the real-time or faster than real-time numerical predictions of adverse space weather events and their influence on the geospace environment put forward higher demands for MHD simulations. Parallel techniques at process level have been widely used to explore scalable high-performance for numerical solutions of MHD equations, such as OpenMP (open multiprocessing) and MPI (Message Passing Interface). With the introduction of general purpose programming frameworks, such as Compute Unified Device Architecture (CUDA) and open Computing Language (openCL), modern graphics processing units (GPUs) can be evolved into a highly capable and low-cost computing solution for scientific research widely. GPUs

* Corresponding author.
E-mail address: fengx@spaceweather.ac.cn (X. Feng).

https://doi.org/10.1016/j.cpc.2018.12.003
0010-4655/© 2018 Published by Elsevier B.V.

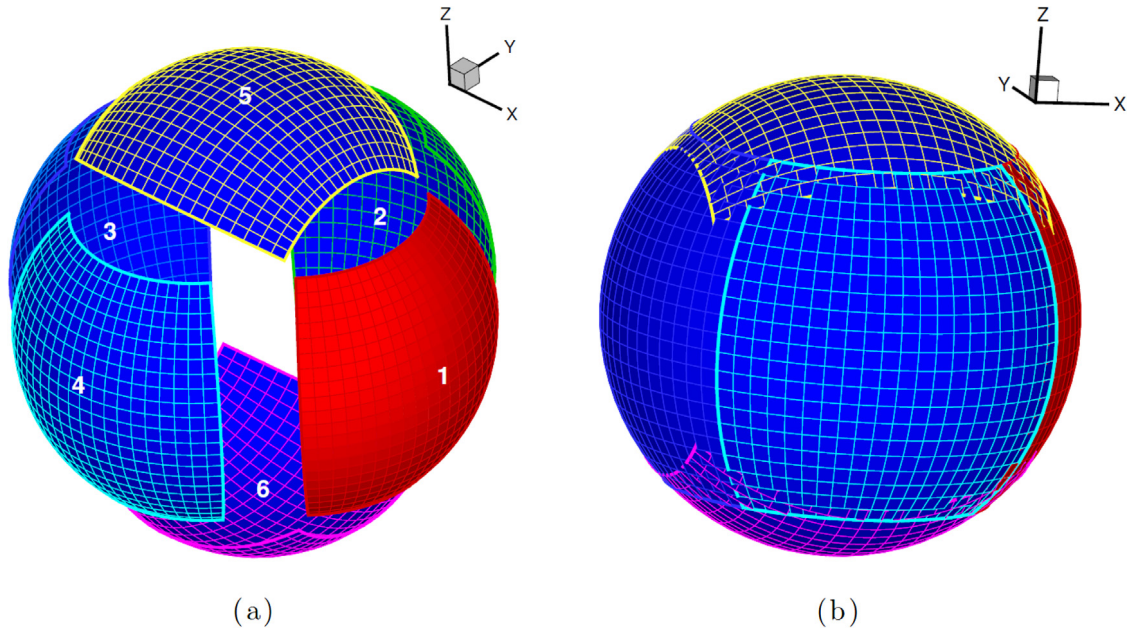

Fig. 1. Basic six-component grid: (a) a spherical overset grid consisting of six components and (b) six identical components are partially overlapped.

devote a larger number of transistors to processing data than CPUs, resulting in better performance for high arithmetic intensity computations. Considering the limitation of memory and calculation units on one graphics card, multiple GPUs are employed for large-scale problems via communication techniques such as MPI, GPU Direct and etcetera.

Some attempts of MHD simulations on GPUs have been presented during the past few years on either a single GPU or multiple GPUs. Wang et al. [15] implemented a compressible inviscid fluid solvers and extended it to support MHD simulations on a NVIDIA Quadro FX 5600 graphics card, acquiring a speedup ratio of 10x than 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, with a close ideal speedup. Wong et al. [16] implemented a total variation diminishing (TVD) algorithm of MHD simulations on a single graphics card (NVIDIA GeForce GTX 295 or GTX480) and achieved a speedup ratio of 84x in 3D than a CPU (Intel Core i7 965 3.2 GHz). Then multi-GPU schemes using MPI and GPUDirect are presented in [17,18] to explore strategies for the data transfer between GPUs that bottlenecked the efficiency of the simulation and improve the peak and average speeds at different resolutions, which achieved peak performances of 97.36 GFLOPS and 2 TFLOPS in double precision. But their works aimed at finite difference (FD) method, which relies on uniform meshes. Lani et al. [19] discussed Object-oriented designs within the COOLFluiD framework [20–22], which is based on a state-of-the-art finite volume (FV) discretization on unstructured meshes and coding techniques by mixing C++ and CUDA, and they acquired a series of speedup ratios ranging 1.51x to 12.32x. Inspired by Lani et al.'s work, we try to develop an easily implemented FV scheme and speed it up with multiple GPU on the six-component grid system. With respect to the domain of spherical shell, as far as I know, only Feng et al. [14, 23] have a trial to implement the Solar-Interplanetary-CESE (SIP-CESE) MHD model [24] with a OpenCL-based GPU programming. But due to the complexity of the model and no optimization, only a speedup ratio of 5x is acquired. To give full play to devices, higher performance should be exploited with some general optimizations on the six-component grid.

In this paper, we first develop a new, spatially second-order accurate finite volume (FV) solver for 3D MHD simulations with the multiple time steps strategy, which is based on the six-component grid for spherical shell domain. Different from previous works [13, 14,23–25] which are also based on the six-component grid, we newly model on hexahedral cells and solve governing equations in a relatively concise form by reducing 3D calculations to 1D with rotation matrix. Then to speed up the simulation, we implement the solver on multiple GPUs with CUDA optimizations and establish an effective multi-GPU FV solver on spherical shell domain. Our computing environments are offered by TH-1HN supercomputer from National Supercomputing Center in Hunan Province. TH-1HN consists of 2048 nodes and each node is equipped with two Intel Xeon E7540 CPUs and one Nvidia M2050 GPU.

Contents are organized as follows: the grid system and decomposition are presented in Section 2. Then a brief description of the FV solver is given in Section 3, including the governing equation and the numerical methods. Section 4 details the implementation of the proposed solver on multiple GPUs. In Section 5, we first show a test case to validate the spacial accuracy of our solver and measure average execution time and speedup ratios of using MPI and multi-GPU techniques, and then apply the solver to the solar wind evolution of a chosen Carrington rotation. Finally, we conclude our works and discuss the future work in Section 6.

2. Grid system and decomposition

2.1. Six-component grid

In order to mitigate the well-known singularity and grid convergence problem at the pole region of the spherical grid, Feng et al. [13] proposed the six-component grid, which consists of six identical grid components to envelope a spherical surface with partial overlap on their boundaries, as shown in Fig. 1. The six grid components have the same shape and size. Each component is just a low latitude region of the usual latitude–longitude grid, which is 90° about the equator and 90° in the longitude. The latitude–longitude boundary value of each component grid can be obtained by interpolation from the neighbor stencils lying in its neighboring component grid. The vector and coordinate defined in different components can be transformed straightforwardly. For an example, the vector field defined in component 1 (i.e. $u_1 = (u_1, v_1, w_1)$) and component 5 (i.e. $u_5 = (u_5, v_5, w_5)$) are related by $u_5 = (-w_1, v_1, u_1)$ and $u_1 = (-w_5, v_5, u_5)$. For more detailed

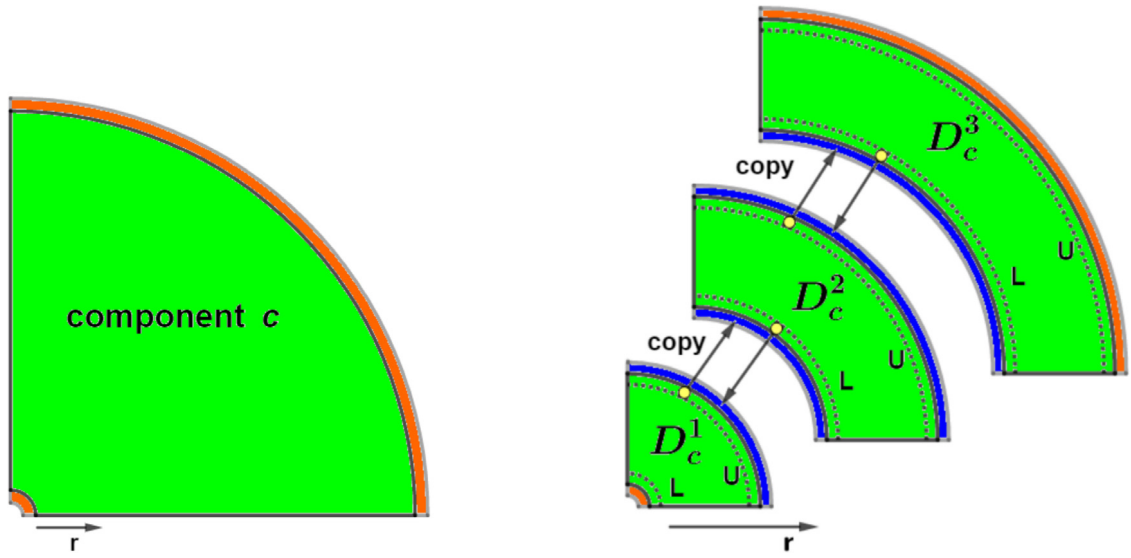

Fig. 2. In the radial direction, the component c are equally divided into subdomains D_c^p, with $p = 1, 2, 3$ referring to each subdomain. Green represents the computation zone, orange represents the boundary zone, and blue represents the ghost zone . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

description of the six-component grid, one may refer to Feng et al. [13].

The six-component grid has been utilized to simulate the evolution of ambient solar wind as well as the solar eruption phenomena [13,14,23–25]. Employing the six-component grid, one can directly apply mathematical methods and numerical codes that have been written for the spherical coordinate system. Efficient and concise programs can be made since the grid components are identical and the vector transformations are simple. However, additional procedures, like vector transformations and interpolations are required for each update.

To be specific, each component grid is defined in the spherical coordinate by

$$(\frac{\pi}{4} - \delta \le \theta \le \frac{3\pi}{4} + \delta) \cap (\frac{3\pi}{4} - \delta \le \phi \le \frac{5\pi}{4} + \delta), \tag{1}$$

where δ is proportionally dependent on the grid spacing entailed for the minimum overlapping area. Each component is confined in the same region as that in Eq. (1), and divisions in $\theta-$ and $\phi-$ directions take place as:

$$\begin{aligned} \theta_j &= \theta_{\min} + \jmath\Delta\theta, \qquad \jmath = -1, 0, \ldots, N_\theta + 2 \\ \phi_\ell &= \phi_{\min} + \ell\Delta\phi, \qquad \ell = -1, 0, \ldots, N_\phi + 2 \end{aligned} \tag{2}$$

and

$$\begin{aligned} \Delta\theta &= (\theta_{\max} - \theta_{\min})/(N_\theta - 2) \\ \Delta\phi &= (\phi_{\max} - \phi_{\min})/(N_\phi - 2)' \end{aligned} \tag{3}$$

where N_θ and N_ϕ are the mesh numbers of the latitude and longitude, respectively. $\theta_{\min} = \frac{\pi}{4}$, $\theta_{\max} = \frac{3\pi}{4}$, $\phi_{\min} = \frac{3\pi}{4}$ and $\phi_{\max} = \frac{5\pi}{4}$. In the case of solar wind simulation for the Sun to interplanetary space, Feng et al. in [13] gave a suggestion of the mesh division in radial direction for 1–25 R_s as: $\Delta r(\imath) = 0.01$ R_s if $r(\imath) < 1.1$ R_s; $\Delta r(\imath) = \min(A \times \log_{10}(r(\imath - 1)), \Delta\theta \times r(\imath - 1))$ with $A = 0.01/\log_{10}(1.09)$ if $r(\imath) < 3.5$ R_s; and $\Delta r(\imath) = \Delta\theta \times r(\imath - 1)$ if $r(\imath) \ge 3.5$ R_s. R_s is the solar radius and also the scaling factor for length. In this way, the discrete or geometrical stiffness caused by disparate mesh cell widths can be mitigated.

2.2. Grid decomposition

Based on the six-component grid, the spherical shell domain is decomposed into six identical components with a resolution of (N_r, N_θ, N_ϕ), and the parallelization can be efficiently and well-balanced implemented in the θ- and ϕ- directions. Besides, each component in the radial direction is equally divided into N_n partitions, as is shown in Fig. 2. Green represents the computation zones, orange represents the boundary zones, and blue represents the ghost zones. The whole computation domain is divided into subdomains D_c^p, with $c = 1, \ldots, 6$ referring to the six components and $p = 1, \ldots, N_n$ referring to each partition in its component. Therefore, the resolution of the computation zone in each subdomain should be $(\frac{N_r}{N_n}, N_\theta, N_\phi)$.

In each subdomain, ghost zones (including boundary zones) are set to surround the computation zone. Two layers of ghost cells are used when calculating cells which are at the edge of the computation zone with the spatially second-order, for the ghost zone ensures that same treatments can be applied on all grid cells in the computation zone. As the ghost cells' locations are beyond the computation zone of their subdomains, which actually locate in the computation zone of their neighbor subdomains, the ghost cells' values can be acquired from the computation zone of their neighbor subdomains. In the right picture of Fig. 2, the ghost cells denoted with '*L*' and '*U*' have the exactly corresponding cells in their lower or upper neighbor subdomains and can be copied directly. Fig. 3 presents the grid decomposition in the θ- and ϕ- directions, and the ghost cells are denoted with '*W*', '*E*', '*S*' and '*N*', respectively. '*W*', '*E*', '*S*' and '*N*' zones can easily be determined by coordinate transformation and interpolation from their west, east, south and north neighbor subdomains.

By utilizing the spherical mesh points generated above, we take their corresponding cartesian coordinates, and form the corresponding hexahedral cells as is shown in Fig. 4. In what follows, we describe the implementation of the governing equations on the hexahedral cells in the Cartesian coordinate.

3. Governing equations and numerical methods

3.1. Ideal MHD equations

The ideal MHD equations consist of a set of nonlinear hyperbolic equations. With the generalized Lagrange multiplier (GLM), we take the conservative form in Cartesian coordinate system as

$$\partial_t \mathbf{U} + \nabla \cdot \mathbf{F} = \mathbf{S} + \mathbf{Q}, \tag{4}$$

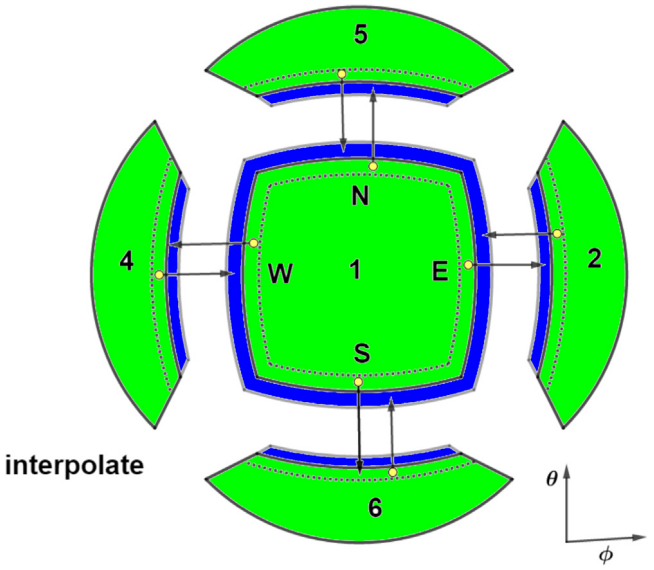

Fig. 3. The grid decomposition in the (θ, ϕ) directions. The ghost zones can easily be determined by coordinate transformation and interpolation from their neighbor subdomains . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

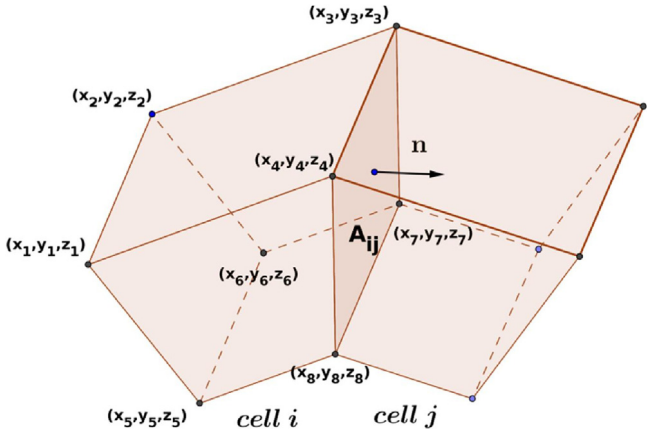

Fig. 4. Hexahedrons used in our solver.

with

$$\mathbf{U} = (\rho, \rho\mathbf{v}, \mathbf{B}, e, \psi)^T \tag{5}$$

and

$$\mathbf{F} = \begin{pmatrix} \rho\mathbf{v} \\ \rho\mathbf{v}\mathbf{v} + (p + \dfrac{\mathbf{B}\cdot\mathbf{B}}{2})\mathbf{I} - \mathbf{B}\mathbf{B} \\ \mathbf{v}\mathbf{B} - \mathbf{B}\mathbf{v} + \psi\mathbf{I} \\ \mathbf{v}(e + p + \dfrac{\mathbf{B}\cdot\mathbf{B}}{2}) - \mathbf{B}(\mathbf{v}\cdot\mathbf{B}) \\ c_h^2\mathbf{B} \end{pmatrix}. \tag{6}$$

$\mathbf{U}$ is the vector of conserved quantities, in which ρ is the mass density, $\mathbf{v} = (v_x, v_y, v_z)$ and $\mathbf{B} = (B_x, B_y, B_z)$ are the velocities and the magnetic field in x-, y-, z- directions respectively, $e = \rho\frac{\mathbf{v}^2}{2} + \frac{p}{\gamma-1} + \frac{\mathbf{B}^2}{2}$ is the total energy and p the pressure. γ is the ratio of specific heats. ψ is introduced by GLM method to couple the divergence constraint $\nabla\cdot\mathbf{B} = 0$ with the conservation laws [26]. In the flux tensor $\mathbf{F}$, $\mathbf{I}$ is the 3×3 identity matrix. c_h and c_p are constant coming up with the GLM method and defined [26,27] as:

$$c_h := \frac{C_{\text{cfl}}}{\Delta t_n} h_{\min}, \quad c_p^2 := \frac{h_{\min} c_h}{\bar{\alpha}} \quad \bar{\alpha} \in [0, 1], \tag{7}$$

and $h_{\min}$ is the minimum height of all cells in every face's normal direction. $\mathbf{S} = (0, \mathbf{0}, \mathbf{0}, 0, -\frac{c_h^2}{c_p^2}\psi)^T$ is the numerical source vector for GLM, and $\mathbf{Q}$ is the physical source vector for specific problems.

3.2. Finite volume formulation

The FV method of Eq. (4) on the hexahedral cells in the six-component mesh grid can be written as

$$\frac{d\mathbf{U}_i}{dt} + \frac{1}{\Omega_i}\sum_{\text{face}_{ij}=1}^{6} \mathcal{R}^{-1}(\mathbf{n}_{ij})\mathbf{F}_{\mathbf{x}}\Big(\mathcal{R}(\mathbf{n}_{ij})\mathbf{U}_i, \mathcal{R}(\mathbf{n}_{ij})\mathbf{U}_j\Big)A_{ij} = \mathbf{S}_i + \mathbf{Q}_i. \tag{8}$$

$\mathbf{U}_i$ refers to the volume-averaged conservative variables of the calculated cell i, and $\mathbf{U}_j$ is that of the corresponding neighbor cell j. As shown in Fig. 4, cells i and j share a common interface face$_{ij}$. Obviously, every cell i has six neighbor cells and six corresponding interfaces, thus we number them as face$_{ij} = 1, 2, \ldots, 6$. A_{ij} is the area of the interface face$_{ij}$, and $\mathbf{n}_{ij}$ is its outer unit normal vector. Ω_i is the volume of cell i. $\mathbf{F}_{\mathbf{x}}$ is the function of the numerical flux term in x-direction. $\mathbf{S}_i$ and $\mathbf{Q}_i$ are volume-averaged source terms. $\mathcal{R}$ is the rotation matrix [26,28] that rotates the x-axis to the direction of $\mathbf{n}_{ij}$ and $\mathcal{R}^{-1}$ rotates it back. In this way, we can consider flux calculations only in x-direction about $\mathbf{F}_{\mathbf{x}}$. The definition of $\mathcal{R}$ and $\mathcal{R}^{-1}$ about unit vector $\mathbf{n}$ is given as

$$\mathcal{R} = \begin{bmatrix} 1 & & & & \\ & \mathbf{r_1} & & & \\ & & \mathbf{r_1} & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \tag{9}$$

with

$$\mathbf{r_1} = \begin{bmatrix} n_x & n_y & n_z \\ t_{1x} & t_{1y} & t_{1z} \\ t_{2x} & t_{2y} & t_{2z} \end{bmatrix}, \tag{10}$$

and

$$\mathcal{R}^{-1} = \begin{bmatrix} 1 & & & & \\ & \mathbf{r_1}^T & & & \\ & & \mathbf{r_1}^T & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}. \tag{11}$$

$\mathbf{t_1} = (t_{1x}, t_{1y}, t_{1z})$ and $\mathbf{t_2} = (t_{2x}, t_{2y}, t_{2z})$ are unit vectors tangent to the surface of the control volume and orthogonal to each other [28].

As for the calculation of the numerical flux term $\mathbf{F}$, a variety of approximate Riemann solvers have been developed during the past decades. Based on the numerical FV method, we not only adopt two most typical solvers, Lax–Friedrichs and Roe [29], but also try a hybrid scheme proposed in [30,31]. This hybrid scheme uses the less dissipative Roe solver for the mainly smooth parts of the flow as well as near contact discontinuities, while the more dissipative Lax–Friedrichs solver is used near strong shocks. The hybrid scheme is as follows,

$$\mathbf{F}_{\text{hybrid}} = 0.5 * (\mathbf{F}(\mathbf{U}_i) + \mathbf{F}(\mathbf{U}_j)) - 0.5 * \mathbf{D}_{\text{hybrid}}, \tag{12}$$

with

$$\begin{aligned} \mathbf{D}_{\text{hybrid}} &= (1-\omega)\mathbf{D}_{\text{Roe}} + \omega\mathbf{D}_{\text{LF}}, \\ \mathbf{D}_{\text{Roe}} &= \mathbf{R}(\mathbf{U}_j - \mathbf{U}_i)|\Lambda|\mathbf{R}^{-1}, \\ \mathbf{D}_{\text{LF}} &= |\lambda_{\max}|(\mathbf{U}_j - \mathbf{U}_i). \end{aligned} \tag{13}$$

$\mathbf{D}_{\text{LF}}$, $\mathbf{D}_{\text{Roe}}$ and $\mathbf{D}_{\text{hybrid}}$ are the dissipation matrices for Lax–Friedrichs, Roe and hybrid schemes. The pressure-based indicator $\omega \in [0, 1]$ is

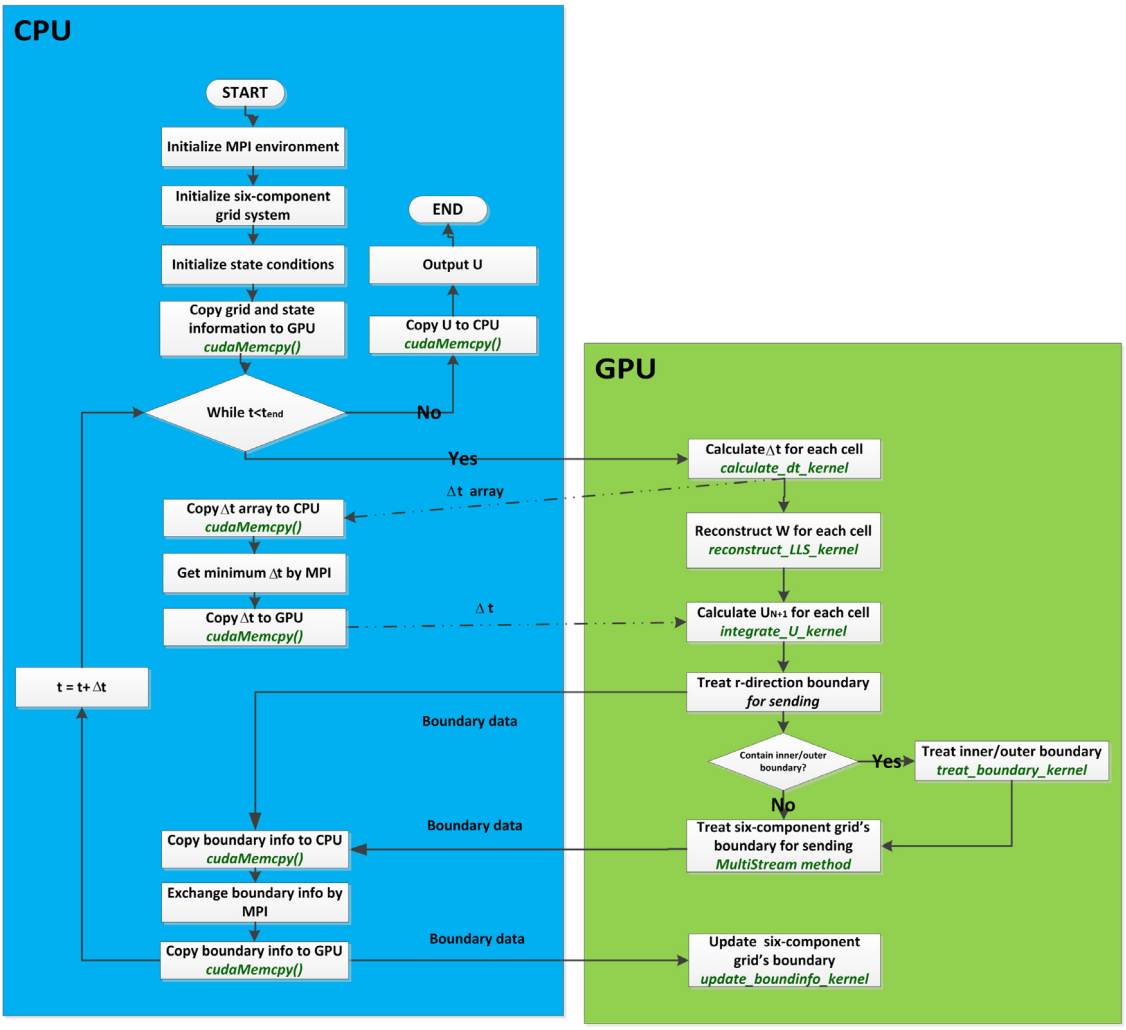

Fig. 5. The program flow chart of the parallel implementation of CPUs and GPUs.

defined as $\omega = \left|\frac{p_i - p_j}{p_i + p_j}\right|^{\frac{1}{2}}$. In $\mathbf{D}_{\text{Roe}}$, $\mathbf{R}$ and $\mathbf{R}^{-1}$ are the mode synthesis matrices of left and right eigenvectors, and Λ is the eigenvalue matrix. $\lambda_{\max}$ in $\mathbf{D}_{\text{LF}}$ is the fastest wave speed evaluated at the interface of cell i and j.

In order to achieve the second-order spatial accuracy, the limited linear least squares reconstruction proposed by Barth [32], which shows good performance in many works [33–35], is employed for our solver. The general formula for the limited reconstruction is used for the primitive variables $\mathbf{W} = (\rho, \mathbf{v}, \mathbf{B}, p, \psi)^T$ on cell i:

$$W_i^{(k)}(\tilde{\mathbf{x}}_r) = \bar{W}_i^{(k)} + \phi_i^{(k)} \nabla W^{(k)} \cdot (\tilde{\mathbf{x}}_r - \tilde{\mathbf{x}}_i). \tag{14}$$

$\tilde{\mathbf{x}}_i$ is the position of cell i's centroid, and $\tilde{\mathbf{x}}_r$ is the position of the point to reconstruct about cell i. Here we usually take $\tilde{\mathbf{x}}_r$ as centroid of cell i's face. $\bar{W}_i^{(k)}$ is the kth component of the primitive vector at $\tilde{\mathbf{x}}_i$. $\nabla W^{(k)}$ is the least-squares gradients, which is calculated using the cell-centered values in neighboring cells, by locally solving a non-square system for the gradient of $W^{(k)}$ by a least-squares approach [33]. Following Ivan's work in [35], we use a 27-point reconstruction stencil in order to provide better robustness against non-uniformity in the grid and solution gradients that are not aligned with the grid. A widely used choice for the slope limiter ϕ_i proposed by Venkatakrishnan [36], is employed, which is believed to not only produce monotone solutions and devoid of oscillations, but keep the accuracy and convergence.

4. Multi-GPU implementation

Based on TH-1HN cluster that each node has two Intel Xeon E7540 CPUs and one Nvidia M2050 GPU, each subdomain is assigned to one node and the computation zone is integrated by its corresponding GPU. After every integration step, ghost cells can be updated by the communication with their neighbor subdomains via MPI. Fig. 5 presents the program flow chart of the parallel implementation of CPUs and GPUs. As described in this flow chart, CPUs control the program's main flow including initialization, data copy between CPUs and GPUs, loop control and MPI communication. While intensive calculation is distributed on GPUs by invoking a series of CUDA kernels.

4.1. Storage of meshes and state variables

We initialize the six-component grid system on the host of each node, including the arrays of mesh information in each subdomain, and then copy them to GPU's global memory. Considering the centroid's coordinate is 3D as (x, y, z), we store every coordinate component in one array separately, for the purpose of avoiding non-unit-stride global memory accesses whenever possible. The 3D meshes are stored as 1D arrays in a contiguous address space, which are allocated by ***cudaMalloc()***. We also use ***cudaBindTexture()*** to bind texture references to these arrays, which enables these arrays to be read through the read-only texture cache by ***tex1Dfetch()***.

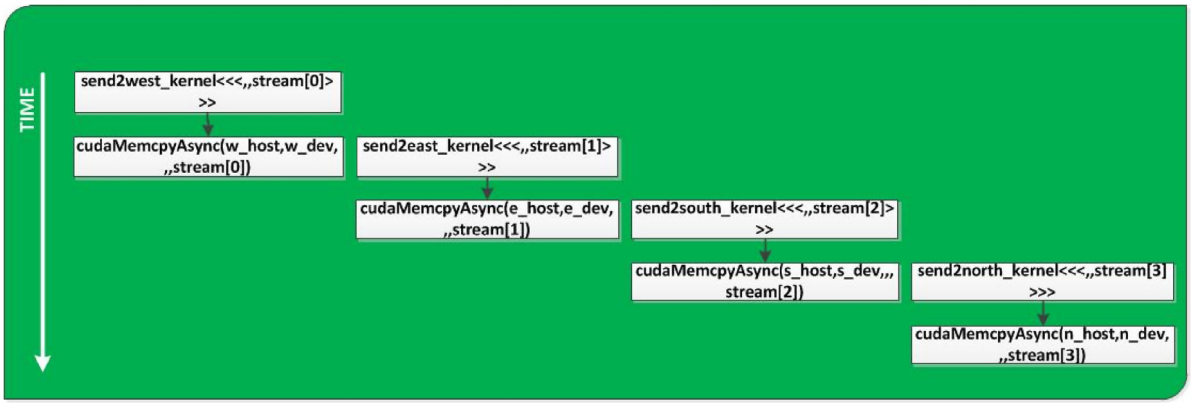

Fig. 6. The exchange information between six-component grid by using multi-stream method.

Primitive variables of each subdomain, (ρ, v_x, v_y, v_z, B_x, B_y, B_z, p, ψ) are initialized on the host, and then copied into global memory. Likewise, nine variables are separately stored in nine 1D arrays. In order to avoid a race condition during integration, we allocate another set of memory, i.e. (ρ^*, v_x^*, v_y^*, v_z^*, B_x^*, B_y^*, B_z^*, p^*, ψ^*) , to separate arrays that are used for reading and writing. Initially, reading pointers $*pin[9]$ are set to the former set of memory, and writing pointers $*pout[9]$ are set to the latter set. After each integration step, state variables pointed by $*pout[9]$ are updated, and then the memory addresses pointed by $*pin[9]$ and $*pout[9]$ are swapped for the next integration step. Besides, the intermediate result arrays of the reconstructed values at facial centroids of each cell are stored in global memory (denoted as *fsArray*[]) in the same manner.

4.2. Primary kernels

As presented in Fig. 5, three primary and most time-consuming kernels are invoked:

- ***calculate_dt_kernel*** calculates the time step Δt;
- ***reconstruct_LLS_kernel*** reconstructs state variables at facial centroids using linear least square method;
- ***integrate_U_kernel*** calculates $\mathbf{U}^{n+1}$ with the FV method.

All these three kernels are implemented as a parallel cell-based loop, in which each cell is mapped onto a GPU thread. A 1D GPU thread grid is used to be in accordance with the arrays of cells (i.e. meshes and state variables), in which each cell is associated with a unique ID. The number of threads per block *BLOCK_SIZE* is defined as 32, and the number of blocks for the kernel is computed by $(N + BLOCK\_SIZE - 1)/BLOCK\_SIZE$, in which N is the local number of mapped cells in the subdomain. Through there is a loop over six facial centroids in each cell in the reconstruction, a parallel cell-based loop should be more effectively than the face-based loop. Because the six facial centroids are calculated on the same 27-cell stencil, and the cell-based ***reconstruct_LLS_kernel*** can deal with the six facial centroids intensively. When it comes to the convective flux of each face, following Lani et al.'s experiment [19] that associating a different thread to each face would have required thread synchronization and undoubtedly affect the overall performance, we choose a cell-based loop for the convective flux as well. As for the source term, it all depends on the cell it belongs to, so a cell-based loop is suitable. Considering these and with the state variables of facial centroids precomputed in ***reconstruct_LLS_kernel***, we set the calculation of the convective flux and source term into ***integrate_U_kernel*** and solve Eq. (8) in each cell. Listing 1 is the pseudo code of ***integrate_U_kernel***, showing how the cells are mapped onto threads and the computation of flux and source term in the cell-based loop. Listing 2 gives a description of the *gridInfo* structure.

As shared memory has much higher bandwidth and lower latency than local and global memory, and can be visited by all the threads in its block, When we seek the minimum Δt among all of the cells, Δt of each cell is calculated by the ***calculate_dt_kernel*** and stored in shared memory as an array *dt_shared*[*blocksize*]. Then half of threads do the comparison between two element of *dt_shared* and update the corresponding shared memory with the preferred value. In the next step, half of them do this again. Repeat this process until a relatively small number of threads are left. Finally, we only copy target data of each block to CPUs for the minimum Δt.

4.3. Boundary treatment and communication

The ghost zone in each subdomain should be updated, after the $\mathbf{U}^{n+1}$ in the computation zone are calculated. Generally, for the ghost cells that have adjacent subdomains, i.e. the blue zones in Figs. 2 and 3, their state values can be found or interpolated in the computation zone of their neighbor subdomains, and acquired by communication via MPI. For the ghost cells that are near the inner and outer boundary of the whole computation domain, i.e. the orange zones in Fig. 2, special treatments are required according to boundary conditions.

In $r-$ direction, for the subdomain that contains orange zones, a cell-based loop over the innermost or outermost layer is utilized by the ***treat_boundary_kernel***. Details of boundary condition will be explained with the tests in Section 5. Blue zones in each subdomain are ghost cells whose corresponding cells can be directly found and copied in neighbor subdomains. To acquire their data from neighbor GPUs, procedures contain GPU-to-host, MPI communication and host-to-GPU.

In $\theta-$ and $\phi-$ direction, as depicted in Fig. 3, ghost cells in blue come from their neighbor subdomains based on the six-component grid. Two steps are required before the communication between each node. First is to interpolate data that are to be sent to west-, east-, south-, north-neighbor by linear least square method in GPUs, and then to copy those data to host for MPI communication. By using multi-stream as depicted in Fig. 6, data sent to west-, east-, south-, north-neighbor are calculated and copied in different streams. In this way, execution of kernels and ***cudaMemcpyAsync*** are task parallel in time and shorten the consuming time.

4.4. Multiple time stepping

At the solar wind background of our research, the plasma density, the alfven velocity, magnetic field, the plasma β and spatial grid size vary greatly with heliocentric distance, implying a large

Listing 1: integrate_U_kernel

```
/*** integrate_U ***/
__global__ void integrate_U_kernel(
 double *pOut[], double *pIn[], double fsArray[], gridInfo grid){
    threadID = (blockIdx.x * blockDim.x) + threadIdx.x;
    N = grid.NCr*grid.NCt*grid.NCp //the number of mapped cells
    if(threadID<N){
        /***map the cell onto the thread***/
        idxr = threadID/grid.NCt*grid.NCp;
        idxt = (threadID i*grid.NCt*grid.NCp)/grid.NCp;
        idxp = threadID i*grid.NCt*grid.NCp j*grid.NCp;
        idxr = idxr+grid.NG;
        idxt = idxt+grid.NG;
        idxp = idxp+grid.NG;
        idxI = idxr*(grid.NTt*grid.NTp)+idxt*gird.NTp+idxp;

        /***compute flux***/
        fluxSum = 0.0
        for(face=1; face<=6; face++){
            getNeighborJ(&idxJ, face,idxr,idxt,idxp);
            getFacialInfo(&arealJ,nIJ, idxI,face);
            getFacialState(&rhoI,&pI,&psiI,vI,B1I,B0I, fsArray,idxI,face);
            getFacialState(&rhoJ,&pJ,&psiJ,vJ,B1J,B0J, fsArray,idxJ,face);
            rotate(vIr,vJr,B0Ir,B0Jr,B1Ir,B1Jr,\
                   vI,vJ,B0I,B0J,B1I,B1J,nIJ);
            computeFlux(fluxIJr, rhoI,pI,psiI,vIr,B1Ir,B0Ir,\
                                 rhoJ,pJ,psiJ,vJr,B1Jr,B0Jr);
            reverse(fluxIJ[face], nIJ,fluxIJr);
            fluxSum += fluxIJ*arealJ
        }

        /***compute source term***/
        getCellState(&rho,&p,&psi,v,B1,B0, idxI, pIn);
        computeSourceTerm(soureTerm, rho,p,psi,v,B1,B0);

        /***integrate state variables***/
        integrateU(pOut, fluxSum,soureTerm,volume,dt)
    }
}
```

Listing 2: gridInfo structure

```
struct gridInfo{
    int NCr; //the number of computational cells in the r direction
    int NCt; //the number of computational cells in the theta direction
    int NCp; //the number of computational cells in the phi direction
    int NG;  //the number of ghost cells' layers
    int NTr; //the total number of cells in the r direction, NTr=NCr+2*NG
    int NTt; //the total number of cells in the theta direction, NTt=NCt+2*NG
    int NTp; //the total number of cells in the phi direction, NTp=NCp+2*NG
    ...
};
```

variation of the CFL stability constraints as well as the time step from the corona to interplanetary space [13]. The multi-time-stepping method was proposed in [37,38] to take different time steps in different parts of the grid with large spatial grid difference, and it can avoid the necessity of taking a single time step in the whole computation domain determined by the numerical CFL stability conditions. Using this method, we can not only keep the numerical simulation steadily forward, but reduce heavily calculation and shorten the computing time. According to [37,39], we divide our domain in the radial direction into areas with A_c^1 in the range of 1–3.5 R_s, A_c^2 in the range of 3.5–10 R_s and A_c^3 in the range of 10–25 R_s ($c = 1, \ldots, 6$ referring to each component), as is shown the left in Fig. 7. We first calculate the time step Δt_c^n for every area A_c^n by using the CFL stability condition and unify Δt^n with the minimum value among six components, i.e. $\Delta t^n = \min_{1\le c\le 6}(\Delta t_c^n)$. Then modify the time step Δt^n ($n > 1$) by

$$M^n = \mathrm{int}(\Delta t^{n+1}/\Delta t^n), \quad \Delta t^{n+1} = M^n \Delta t^n$$

($n = 1, 2$ successively).

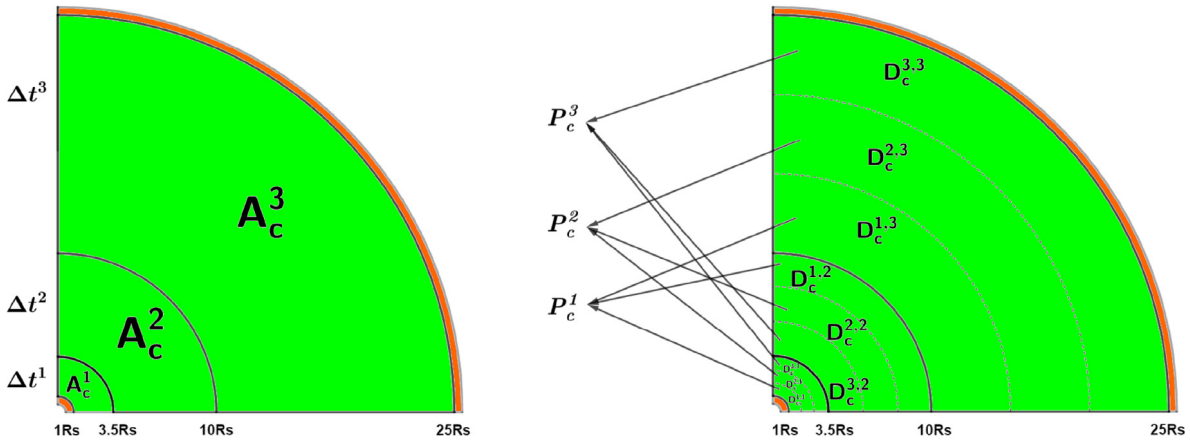

Fig. 7. Time stepping method with the computation domain division.

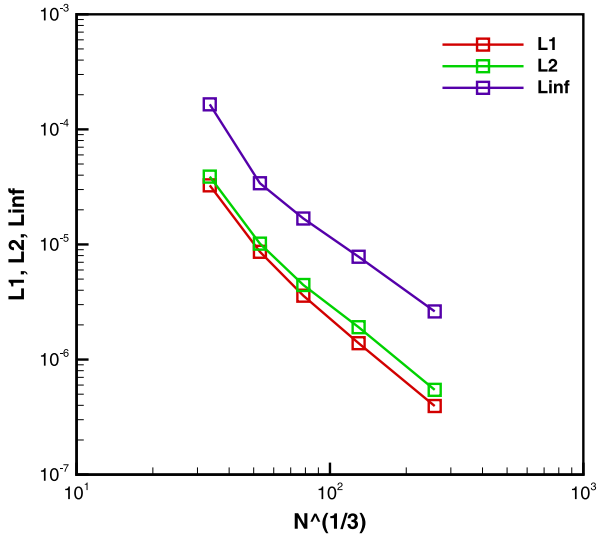

Fig. 8. L_1, L_2 and L_∞ error norms in density for the manufactured MHD solution.

When every A_c^n calculates M^n loops with Δt^n, the A_c^{n+1} advances a single step of Δt^{n+1}. After A_c^1 advancing $M^2 \times M^1$ steps of Δt^1, A_c^2 advancing M^2 steps of Δt^2, and A_c^3 advancing a step of Δt^3, all the three areas reach the same time level and another set of time steps Δt^n for each area A_c^n is determined again.

Well-balanced calculation's load should also be considered with the multiple time stepping method. Directly distributing every area A_c^n with one or more processors has been used in the present works [13,25]. Even though we can manually adjust the allocation of computing resources on every areas A_c^n due to the different calculation loops, it is still difficult to generally balance all the processors. This is because that every area may have a different number of cells and mesh decomposition is hard to decide. What is more, with the calculation advancing, Δt^n varies and M^n are unsure in advance. So we proposed another strategy as is shown the right in Fig. 7. Every area is equally divided in the radial direction into a same number of subdomains, according to the mesh decomposition described in Section 2.2. Here we denote subdomains as $D_c^{p,n}$ with c meaning the component, n meaning the area and p meaning the partition number in the area. Then every processor P_c^p is allocated with subdomains $D_c^{p,1}$ $D_c^{p,2}$ and $D_c^{p,3}$, meaning that every processor calculates a part of areas A^1, A^2 and A^3. In this way all of the processors are equally loaded and are fully utilized with as short computing time as possible. It should be noticed that in A_c^2, the order of the subdomains is contrary to that in A_c^1 and A_c^3, to avoid some data transfers of the boundary of areas between GPUs.

5. Numerical results

5.1. MHD manufactured solution

To validate the grid system for spherical shell and the proposed multi-GPU FV solver, we adopt the test case of a 3D steady-state axi-symmetric solution of a MHD plasma on a spherical shell domain flowing outward at superfast speeds, proposed by Ivan et al. in [35]. Average execution time and speedup ratios of using multi-GPU techniques are also measured in this case. We should notice here that, in this paper, every test program is run for five times and the average execution time is measured. The exact solution is specified with the primitive variables as:

$$\mathbf{W} = (R^{-\frac{5}{2}}, \frac{x}{\sqrt{R}}, \frac{y}{\sqrt{R}}, \frac{z}{\sqrt{R}} + \kappa R^{\frac{5}{2}}, \frac{x}{R^3}, \frac{y}{R^3}, \frac{z}{R^3} + \kappa, R^{-\frac{5}{2}}, 0)^T, \tag{15}$$

with a source term added to the ideal MHD equations, i.e.

$$\mathbf{Q} = \begin{pmatrix} 0 \\ \frac{1}{2}xR^{-\frac{5}{2}}(R^{-1} - 5R^{-2} - \kappa z) \\ \frac{1}{2}yR^{-\frac{5}{2}}(R^{-1} - 5R^{-2} - \kappa z) \\ \frac{1}{2}zR^{-\frac{5}{2}}(R^{-1} - 5R^{-2} - \kappa z) + \frac{5}{2}R^{-\frac{1}{2}}\kappa(1 + \kappa Rz) + \kappa R^{-\frac{1}{2}} \\ \vec{0} \\ \frac{1}{2}R^{-2} + \kappa z(3.5R^{-1} + 2\kappa z) + \frac{(\kappa R)^2}{2}(7 + 5\kappa Rz) \\ 0 \end{pmatrix}. \tag{16}$$

The ratio of plasma specific heat is $\gamma = 1.4$, and the perturbation parameter is taken as $\kappa = 0.017$ to keep the solution significant in latitudinal variation and the flow supersonic in the domain. The magnetic field is irrotational and aligned with the velocity such that $\mathbf{v} \times \mathbf{B} = 0$ everywhere. The computation domain in this case is defined by inner and outer spheres of radius $R_i = 2$ and $R_o = 3.5$, respectively. The inner boundary condition is specified based on the exact solution while the outer uses linear extrapolation.

In Fig. 8, the L_1, L_2 and L_∞ error norms in density are obtained on a series of mesh divisions with the resolution of each component

Table 1
Average total execution time and speedup ratios of 1000 time-step loops on GPUs and CPUs.

Mesh resolution per component	6 GPUs (s)	6 CPU processes (s)	Speedup (x)
$21 \times 62 \times 62$	55.80	1571.52	28.16
$36 \times 102 \times 102$	245.64	7026.66	28.61
Mesh resolution per component	18 GPUs (s)	18 CPU processes (s)	Speedup (x)
$36 \times 102 \times 102$	93.40	2594.05	27.77
$71 \times 202 \times 202$	635.45	19,099.30	30.06

Table 2
Average total execution time and speedup ratios of 1000 time-step loops on GPUs and CPUs, with cell number and device number in proportion.

Mesh resolution per component & device number	GPUs (s)	CPU processes (s)	Speedup (x)
$36 \times 102 \times 102$ & 6	245.64	7026.66	28.61
$72 \times 102 \times 102$ & 12	248.09	7330.07	29.55
$108 \times 102 \times 102$ & 18	249.83	7356.97	29.45

(N_r, N_θ, N_ϕ) ranging from $13 \times 22 \times 22$ to $71 \times 202 \times 202$, namely from 37752 to 17382504 cells in total. With refinement of the grids, the lines of L_1, L_2 and L_∞ norms have slopes of about -2.1601308477, -2.0872626565 and -2.0271203117, respectively. This implies that the second-order theoretical accuracy of the solver are achieved in all the error norms for smooth but non-radial flows with a magnetic field, and validates the specified spatial discretization procedure on the six-component grid.

Table 1 presents the average total execution time of 1000 time-step loops on GPUs and CPUs, and the speedup ratios between them. Mesh resolutions of $21 \times 62 \times 62$, $36 \times 102 \times 102$ and $71 \times 202 \times 202$ per component, corresponding to 484 344, 2 247 264 and 17 382 504 cells in total, are chosen for general measurements. By employing 6 GPUs and 6 CPU processes respectively, we get speedup ratios of 28.16x on the mesh of $21 \times 62 \times 62$ per component, and 28.61x on the mesh of $36 \times 102 \times 102$. When comparing 18 GPUs and 18 CPU processes, the formers are 27.77x and 30.06x times faster than the latters, on the meshes of $36 \times 102 \times 102$ and of $71 \times 202 \times 202$, respectively. With the same number of GPUs, we can easily see that the denser the mesh is, the more significant speedup ratio is achieved.

To validate the scalability of the solver on GPUs and CPUs, we measure the average total execution time and the speedup ratios of 1000 time-step loops on GPUs and CPUs, with the cell number and device number in proportion, as is shown in Table 2. The speedup ratios keep almost the same when the processing unit number goes up, along with the problem size, which shows very nice scalability.

Table 3 presents the average execution time of primary computation modules with the mesh resolution of $36 \times 102 \times 102$ per component on 6, 12 and 18 GPUs, respectively. We can easily find that the ***reconstruct_LLS_kernel*** is most time-consuming and accounts for more than 60% on 6, 12 and 18 GPUs, which is due to its calculation complexity. Compared with the case of 6 GPUs , the average execution time of ***calculate_dt_kernel***, ***reconstruct_LLS_kernel*** and ***integrate_U_kernel*** on 12 GPUs gets speedup ratios of 1.91x, 1.89x and 1.91x respectively. While the speedup ratios of them on 18 GPUs are 2.96x, 2.73x and 2.95x respectively. The speedup ratios are in relatively accordance with the number of GPU and are reasonable. Besides, we also notice that, on both 12 and 18 GPUs, the speedup ratios of the ***reconstruct_LLS_kernel*** are a little lower than those of ***calculate_dt_kernel*** and ***integrate_U_kernel***. This is because that, in the ***reconstruct_LLS_kernel***, not only the computation zone but a layer of ghost cells should be mapped onto threads, and extra calculation will affect the speedup ratio. By using multistream method, the treatment of six-component boundary costs 13.35 ms on 6 GPUs, 6.87 ms on 12 GPUs and 5.43 ms on 18 GPUs. Other module contains copying boundary data from GPU to the host, MPI communications between different nodes, copying boundary data back to GPU and some trivial operations. It takes 7.33 ms, 7.30 ms and 7.31 ms in every case and correspondingly accounts for 2.98%, 5.0% and 7.83% of their total time, respectively. It causes extra overheads on distributed multi-GPU systems, and is difficult to proportionally reduce. The average execution time of this module is almost the same, but with the increment of GPUs, the communication overhead between different nodes accounts more and will become the bottleneck of speedup ratios. Only when the number of computing devices matches with data granularity, can we get a considerable and worthy speedup ratio on the whole. In this view, total speedup ratios of 1.85x and 2.63x with 12 and 18 GPUs are reasonable.

5.2. Solar wind simulation

We test this solver by solving the solar wind evolution, which is governed by the modified MHD equations. When solving the MHD equations in a (near) conservation form, it is very important to split the magnetic field $\mathbf{B}$ into a time-independent potential magnetic field $\mathbf{B_0}$ and a time-dependent deviation $\mathbf{B_1}$, i.e. $\mathbf{B} = \mathbf{B_0} + \mathbf{B_1}$ [13,19,40,41]. The reason is that the total energy density can be completely dominated by the magnetic energy ${\mathbf{B_0}}^2/2$ near the Sun, which can lead to negative pressure if calculated from the total energy density. Splitting $\mathbf{B}$ can mitigate this problem effectively near the Sun. As for small plasma β regions, it is inherently more accurate to solve for the deviation $\mathbf{B_1}$ from the embedded field $\mathbf{B_0}$ than to solve for the full magnetic field vector $\mathbf{B}$. The source term is defined as

$$\mathbf{Q} = \begin{pmatrix} 0 \\ \mathbf{j}_0 \times \mathbf{B}_0 + \rho\left[\mathbf{g} - \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \mathbf{r})\right] - 2\rho\boldsymbol{\Omega} \times \mathbf{v} + \mathbf{S}_m \\ -\dfrac{\partial \mathbf{B}_0}{\partial t} \\ -\mathbf{B}_1 \cdot \dfrac{\partial \mathbf{B}_0}{\partial t} + \mathbf{E} \cdot \mathbf{j}_0 + \rho\mathbf{v} \cdot \left[\mathbf{g} - \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \mathbf{r})\right] + Q_e + \mathbf{v} \cdot \mathbf{S}_m \\ 0 \end{pmatrix} \tag{17}$$

with $\mathbf{E} = \mathbf{v} \times \mathbf{B}$, $\mathbf{j}_0 = \nabla \times \mathbf{B}_0$. t and $\mathbf{r}$ are time and position vector originating at the center of the Sun, $\mathbf{g} = -GM/r^3 \cdot \mathbf{r}$ is the solar gravitational force, $\boldsymbol{\Omega}$ is the solar angular speed and $\gamma = 1.05$ is the ratio of specific heats. ρ, $\mathbf{v}$, p, $\mathbf{B}$, $\mathbf{r}$, t, and $\mathbf{g}$ are normalized by the characteristic values ρ_S, a_0, $\rho_S a_0^2$, $\sqrt{\rho_S a_0^2}$, R_S, R_S/a_0, and a_0^2/R_S, where ρ_S, a_0 and R_S are the density, sound speed on the solar surface and solar radius. The solar rotation is considered in the present study with angular velocity $|\boldsymbol{\Omega}| = 2\pi/25.38$ radian day^{-1} (here normalized by a_0/R_S). $\mathbf{S}_m$ and Q_e stand for the momentum and energy source terms, which are responsible for acceleration and heating of the solar wind. Taking into account the magnetic field topology effects [42] with a expansion factor "f_S" in them, the volumetric heating function and momentum source term are

$$\mathbf{S}_m = M\left(\frac{r}{R_S} - 1\right)\exp\left(-\frac{r}{L_M}\right) \cdot \mathbf{r}/r, \tag{18}$$

$$Q_e = Q_1 \exp\left(-\frac{r}{L_{Q_1}}\right) + Q_2\left(\frac{r}{R_S} - 1\right)\exp\left(-\frac{r}{L_{Q_2}}\right). \tag{19}$$

Details about those parameters are described by Feng et al. in [13].

To test the GPU-solver's capability, we numerically study the ambient solar wind of Carrington rotation (CR) 2060 in the descending phase of solar cycle 23 lasting from August 21 to September 17 in 2007. As the solar activity is relatively low in this period and the solar corona has a typical and characteristic structure, CR 2060 is a preferred choice and has been studied in many researches [43,44]. We utilize the line-of-sight photospheric

Table 3
Average execution time and speedup ratios of primary computation modules on GPUs, with the mesh resolution of $36 \times 102 \times 102$ per component.

Computation module	6 GPUs		12 GPUs		18 GPUs	
	Time (ms)	Percentage	Time (ms)	Speedup (x)	Time (ms)	Speedup (x)
calculate_dt_kernel	8.70	3.54%	4.55	1.91	2.94	2.96
reconstruct_LLS_kernel	162.71	66.25%	86.11	1.89	59.58	2.73
integrate_U_kernel	53.51	21.79%	28.01	1.91	18.11	2.95
Treat six-component boundary	13.35	5.44%	6.87	1.94	5.43	2.46
Others	7.33	2.98%	7.30	1.0	7.31	1.0
Total	245.60	100.00%	132.84	1.85	93.40	2.63

Table 4
Average execution time and speedup ratios of the CR 2060 solar wind simulation to reach a steady state.

Mesh resolution per component	18 GPUs (h)	18 CPU processes (h)	Ratio of speedup (x)
$138 \times 62 \times 62$	9.39	268.64	28.61
$210 \times 102 \times 102$	36.71	1074.66	29.28

magnetic data from the Global Oscillation Network Group (GONG) program to produce a 3D global magnetic field in the computation domain with the potential field (PF) model, and specify it as the time-independent $\mathbf{B_0}$. $\mathbf{B_1}$ is initially set as zero. The initial distributions of plasma density ρ, pressure p and velocity $\mathbf{v}$ are given by Parker's solar wind flow [45]. The temperature and number density on the solar surface are $T_S = 1.3 \times 10^6$ K and $\rho_S = 1.5 \times 10^8$ cm^{-3} respectively. With the spherical shell domain, two boundaries should be designated: the inner boundary of the solar surface at 1 R_s, and the outer boundary at 25 R_s. As the six-component grid is of sphere-surface body-fitting, it is easy to implement the inner boundary conditions at the solar surface. For the steady simulation of the ambient solar wind, the inner boundary is fixed. Since the outer boundary in interplanetary is a supersonic/super-Alfvenic region, state variables at the outer boundary are simply extrapolated to boundary ghost cells in the radial direction.

5.2.1. Time measurement

We adopt the six-component grid with the mesh resolution of $138 \times 62 \times 62$ and $210 \times 102 \times 102$ per component, covering the space from the Sun's surface to beyond 20 solar radius spherical surface, thus offer a model for the research of corona activities. The average execution time of simulations is compared between 18 GPUs and 18 CPU processors.

We measure the calculation time that the background solar wind simulation need to reach a steady state for CR 2060. As is shown in Table 4, it need about 268.64 h (about 11.19 days) and 1076.66 h (about 44.7 days) to finish these work on 18 CPU processors, but only 9.39 h and 36.71 h with GPUs. And speedup ratios of 28.61x and 29.28x are achieved by 18 GPUs on the six component grid, with the mesh resolution of $138 \times 62 \times 62$ and $210 \times 102 \times 102$ per component (3,182,832 and 13,109,040 meshes in total), which are our frequently used mesh division schemes. The average execution time is sharply reduced to achieve real-time simulation.

In the previous work by Feng et al. [14,23], they measured the execution time on 24 GPUs (including 10 T C1060 and 14 T C2050) and on same CPU processes. They acquire a speedup ratio of 5x without any GPU optimization. However, by taking into account of some general GPU optimizations as well as the load balance consideration, we achieve relatively considerable speedup ratios on 18 T M2050 GPUs. And as we have demonstrated in Section 5.1 that the solver shows very nice scalability, the speedup ratio will maintain when the number of GPUs goes up along with the problem size.

5.2.2. Comparisons with observations

For the purpose of validating the simulated results, we compare them with the available coronal observations. Some distinct features include multiple coronal streamers, heliospheric current sheets (HCSs) with significantly high inclination and sparser and cooler fast solar wind, and temporal profiles of the radial speed and radial magnetic field polarities are analyzed with these comparisons.

Synoptic maps at 2.6 R_s Fig. 9 presents the synoptic maps at 2.6 R_s for CR 2060. Pictures in the top row show the maps of white-light polarized brightness (pB) at the east and west limbs observed by SOHO/LASCO-C2, with the bright areas in pB images often indicating high-density structures near the sky plane along the line of sight through these points. In the bottom row, the MHD simulated number density N and radial speed Vr are presented with units of 10^5 cm^{-3} and km s^{-1} respectively. The black lines denote the magnetic neutral lines from the MHD model, while white lines from the PFSS model. We can see that the magnetic neutral lines calculated from both the MHD and PFSS models are surrounded by the bright white-light structures at the east and west limbs, which are identified as the streamer belts. The locations of the bright structures in the white-light pB images are characterized by the relatively low velocity and high plasma density in the simulation, while the dark regions correspond with the locations of the increased flow speed and decreased plasma density. Two characteristically warped structures in the streamer belt, located between $\phi = 90° - 200°$ and $\phi = 200° - 260°$ in mid- and low-latitude, produce the dark regions in white-light images at the east and west limbs, which is spatially coincident with the low-density high-speed flow.

Comparison at meridian plane In Fig. 10, we compare the results on meridian planes at $\phi = 180° - 0°$ from observations and simulations to further demonstrate that our MHD solution can describe the specific coronal observations. The left presents the coronal images from 2.3 to 6 R_s observed by SOHO/LASCO-C2, which are taken at the seventh day during rotations, while the right presents the pB images of the simulated results from 1.5 to 6 R_s, which correspond to the same plane with the left. We can find that these streamer-like structures do not extend radially outward from their foot points, but cover relatively large latitudes near the Sun [43]. In the observatory picture, the brightest and sharpest two structures are observed on the right limb, covering the latitude from 30°N to 45°S. On the left limb, there are two bright structures at the latitude of 40°N and 40°S. All the four obvious streamer-like structures are captured at almost the same latitudes in simulated results, though streamers appear shorter.

Fig. 11 shows the simulated steady solar coronal solution at the plane of $\phi = 180° - 0°$ (top row) and $\phi = 270° - 90°$ (bottom row). The color contours stand for the radial speed Vr and number density N, while streamlines denote the magnetic field lines. The magnetic field lines at high latitudes extend into interplanetary space and the solar wind in this region has relatively high velocity and low density. While the low velocity and high density solar wind is located at lower latitudes around the HCS.

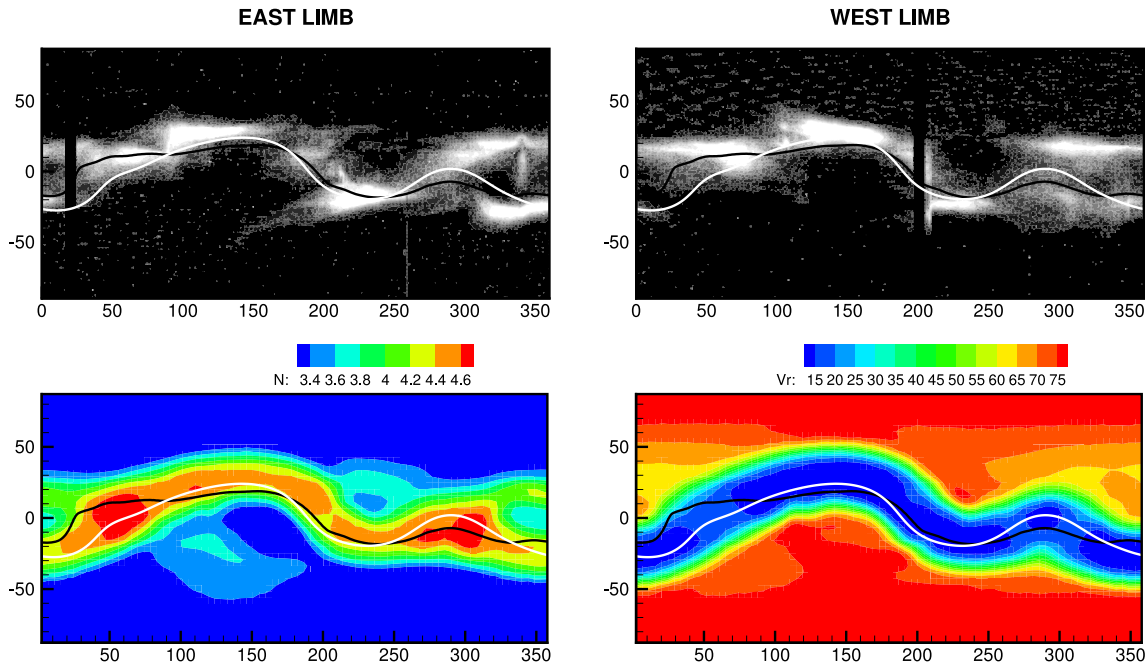

Fig. 9. Synoptic maps at 2.6 R_s for CR 2060. In the top row are the white-light pB observations at east and west limbs from SOHO LASCO C2, and in the bottom row are simulated number density N (unit: 10^5 cm^{-3}) and radial velocity (unit: km s^{-1}). Black lines denote the magnetic neutral lines from MHD models, while white lines from PFSS models.

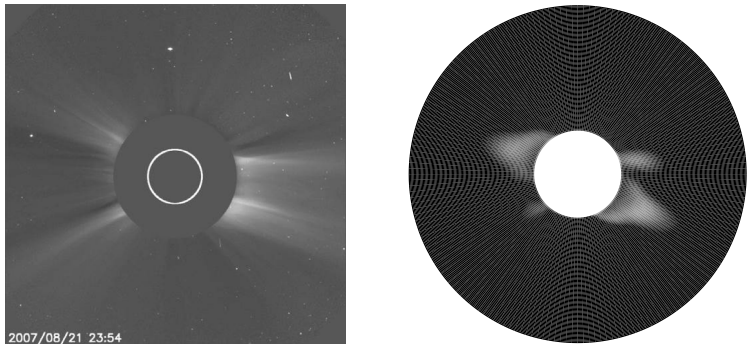

Fig. 10. The coronal observation and simulated result on the meridional plane at $\phi = 180° - 0°$ for CR 2060. The left is the coronal image from 2.3 to 6 R_s observed by SOHO/LASCO-C2, and the right is the pB image of simulated result from 1.5 to 6 R_s.

Comparison with OMNI data at 20 R_s As our simulation ranges from 1 R_s to more than 20 R_s, we map the interplanetary measurements back to 20 R_s by using a ballistic approximation, in which the variations in longitude are computed from the time interval required for a plasma parcel traveling from 20 R_s to the spacecraft location with the in-situ measured solar wind velocity [43]. The interplanetary observational data are obtained from the GSFC/SPDF OMNIWeb interface at http://omniweb.gsfc.nasa.gov. Temporal profiles of the radial solar wind speed and the radial magnetic field polarities from the mapped observational results of the OMNI data (black lines or diamonds) at 20 R_s and the simulated results (red lines) are presented. We can see from Fig. 12 that basically the same trend is followed by the profiles of the solar wind from both the in-situ observation and the simulated result. In the temporal profile of the radial solar wind speed on the left panel, the most distinct two high-speed streams at Longitudes 120° and 230° are precisely captured by the simulated result, at the speed of about 650 km s^{-1}, and the small trough near the first high-speed stream is even captured at Longitude 125°. This implies that our model can accurately simulate the arriving time of high-speed streamers. However, it also contains some minor discrepancies. The trough at Longitude 100° is missed. The simulated low-speed streamer at Longitude 295° is about 400 km s^{-1}, which is higher than the observed 300 km s^{-1}. As far as the polarities of the radial magnetic field are concerned, we can see in the right panel in Fig. 12 that these different sectors simulated by our model are roughly in accordance with that from the observations, even though the crossing longitudes are not well aligned with small deviations. There are a few errors as well, and this is because the waves and perturbations in the field can lead the opposite polarity to be measured rather than the true field polarity [43,46]. In order to quantitatively demonstrate the simulating ability of our model, we define the hit ratio as the ratio that the simulated polarities are the same with the corresponding mapped observational ones, and the hit ratio of CR 2060 is 74.03%.

6. Conclusion

In this paper, we develop a new, spatially second-order accurate finite volume (FV) solver for the 3D MHD equations, which is based on the six-component grid for the spherical shell domain. And then, we implement it on GPUs to speedup the simulation. Performances

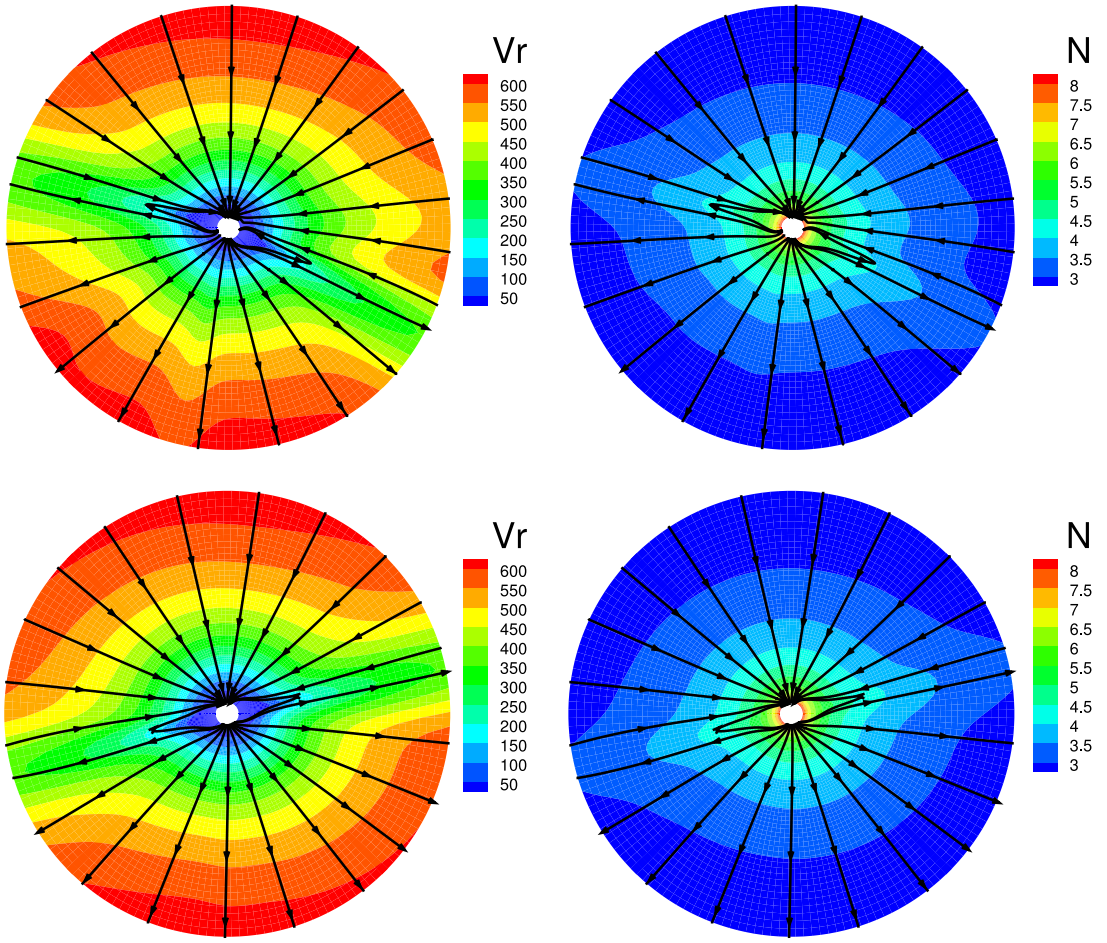

Fig. 11. Solar coronal solution for radial velocity Vr (unit: km s^{-1}), number density N (unit: log_{10} cm^{-3}) and magnetic fields at $\phi = 180° - 0°$ (top row) and $\phi = 270° - 90°$ (bottom row), ranging from 1 to 20 R_s.

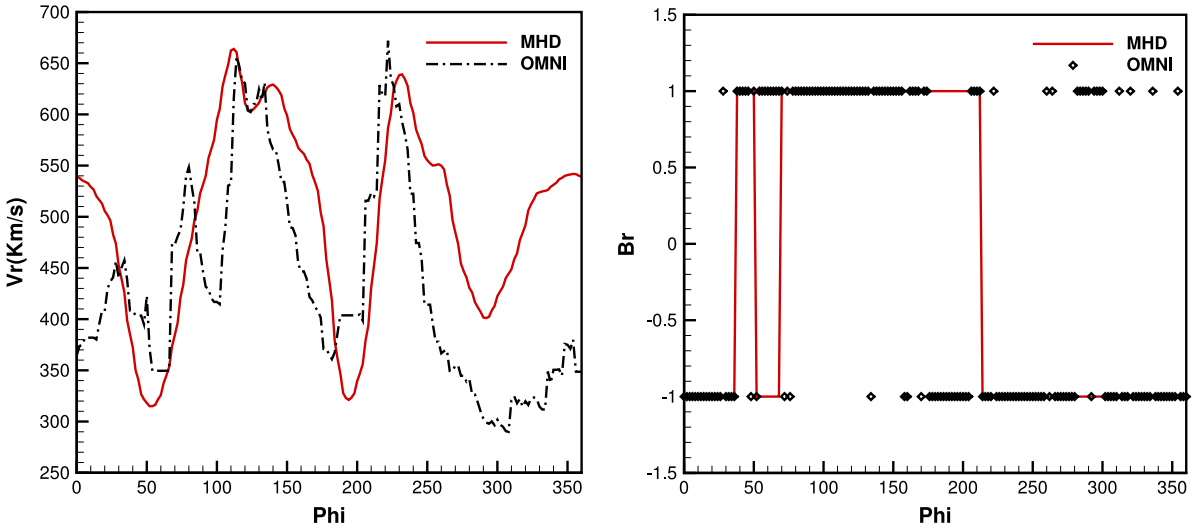

Fig. 12. Comparisons at 20 R_s between the mapped measurements from OMNI data (black lines or diamonds) and the models' results (red lines) for CR 2060. The left is the temporal profile of the radial speed Vr with unit: km s^{-1}, and the right is the profile of the radial magnetic field polarities, with "+1" standing for the radial magnetic field away from the Sun and "−1" towards the Sun . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of them are measured and compared in this work. Firstly, in the MHD manufactured solution problem, we validate that the expected second-order accuracy has been achieved. Speedup ratios of 27x to 30x are obtained by a series of performance's comparisons between CPUs' and GPUs' implementation, which show GPUs' advantages in those high arithmetic intensity calculation. Besides, we conclude that both solvers have nice scalability, but communication overhead between different nodes will become the bottleneck of speedup ratio with the increasement of GPUs, and should match the increased number of GPUs with data granularity

to get a considerable and worthy speedup. Then we apply both solvers to solar wind simulation from the Sun's surface to more than 20 R_s spherical surface. The multi-GPU FV solver obtains speedup ratios of about 29.0x. From comparisons of the simulated results and the observations about CR 2060, we can see that the multi-GPU FV solver can basically capture the features of the multiple coronal streamers, HCSs and temporal profiles of the high and low speed solar wind. It shows roughly agreement in the shapes and distributions of corona between simulations and observations, and implies that the multi-GPU FV solver has good suitability and high efficiency in the study of corona.

In future work, the effective multi-GPU FV solver will be devoted to other topics in solar-terrestrial numerical studies, such as the numerical modeling of the solar disturbance propagation in the solar corona and the solar wind background from the Sun to Earth. Besides, optimizations of GPUs' implementation will be further explored.

Acknowledgments

The work is jointly supported by the National Natural Science Foundation of China (Grant Nos. 41531073, 41731067, 41574171, and 41874202) and the Specialized Research Fund for State Key Laboratories. This work utilizes data obtained by the Global Oscillation Network Group (GONG) program, managed by the National Solar Observatory, which is operated by AURA, Inc. under a cooperative agreement with the National Science Foundation. The SOHO/LASCO data used here are produced by a consortium of the Naval Research Laboratory (USA), Max-Planck-Institut fuer Aeronomie (Germany), Laboratoire d'Astronomie (France), and the University of Birmingham (UK). The OMNI data are obtained from the GSFC/SPDF OMNIWeb interface at http://omniweb.gsfc.nasa.gov. The computing environments of this work are offered by TH-1HN supercomputer from National Supercomputing Center in Hunan Province.

References

[1] A.V. Usmanov, M. Dryer, Solar Phys. 159 (1995) 347–370.
[2] S.T. Wu, W.P. Guo, D.J. Michels, L.F. Burlaga, J. Geophys. Res. 104 (1999) 14789–14801.
[3] J.A. Linker, Z. Mikic, D.A. Biesecker, R.J. Forsyth, S.E. Gibson, A.J. Lazarus, A.R. Lecinski, P. Riley, A. Szabo, B.J. Thompson, J. Geophys. Res. 104 (1999) 9809–9830.
[4] D. Odstrcil, J.A. Linker, R. Lionello, Z. Mikic, P. Riley, V.J. Pizzo, J.G. Luhmann, J. Geophys. Res. 107 (2002).
[5] X.S. Feng, C.Q. Xiang, D.K. Zhong, Q.L. Fan, Chin. Sci. Bull. 50 (2005) 672–678.
[6] K. Hayashi, Astrophys. J. Suppl. Ser. 161 (2005) 480.
[7] F. Shen, X.S. Feng, S.T. Wu, C.Q. Xiang, J. Geophys. Res. 112 (2007).
[8] P.L. Israelevich, T.I. Gombosi, A.I. Ershkovich, K.C. Hansen, C.P.T. Groth, D.L. Dezeeuw, K.G. Powell, Astron. Astrophys. 376 (2001) 288–291.
[9] T. Tanaka, J. Geophys. Res. 108 (2003).
[10] X.S. Feng, Y.F. Zhou, S.T. Wu, Astrophys. J. 655 (2007) 1110.
[11] J. Kleimann, A. Kopp, H. Fichtner, R. Grauer, Ann. Geophys. 27 (2009) 989–1004.
[12] C. Ronchi, R. Iacono, P.S. Paolucci, J. Comput. Phys. 124 (1996) 93–114.
[13] X.S. Feng, L.P. Yang, C.Q. Xiang, S.T. Wu, Y.F. Zhou, D.K. Zhong, Astrophys. J. 723 (2010) 300.
[14] X.S. Feng, D.K. Zhong, Y. Xiang, C. Q. Zhang, Sci. China Earth Sci. 56 (2013) 1864–1880.
[15] P. Wang, T. Abel, R. Kaehler, New Astronomy 15 (2010) 581–589.
[16] H.-C. Wong, U.-H. Wong, X.S. Feng, Z. Tang, Comput. Phys. Comm. 182 (2011) 2132–2160.
[17] U.-H. Wong, H.-C. Wong, Y. Ma, Comput. Phys. Comm. 185 (2014a) 144–152.
[18] U.-H. Wong, T. Aoki, H.-C. Wong, Comput. Phys. Comm. 185 (2014b) 1901–1913.
[19] A. Lani, M.S. Yalim, S. Poedts, Comput. Phys. Comm. 185 (2014) 2538–2557.
[20] A. Lani, T. Quintino, D. Kimpe, H. Deconinck, S. Vandewalle, S. Poedts, International Conference on Computational Science, Vol. 3514, 2005, pp. 279–286.
[21] A. Lani, T. Quintino, D. Kimpe, H. Deconinck, S. Vandewalle, S. Poedts, Sci. Program. 14 (2006) 111–139.
[22] A. Lani, N. Villedie, K. Bensassi, L. Koloszar, M. Vymazal, S.M. Yalim, M. Panesi, 21st AIAA Computational Fluid Dynamics Conference, p. 2589.
[23] X.S. Feng, D.K. Zhong, C.Q. Xiang, Y. Zhang, et al., Numerical Modeling of Space Plasma Flows (Astronum 2012), ASP Conf. Ser, Vol. 474, pp. 131–139.
[24] X.S. Feng, L.P. Yang, C.Q. Xiang, C.W. Jiang, X.P. Ma, S.T. Wu, D.K. Zhong, Y.F. Zhou, Solar Phys. 279 (2012) 207–229.
[25] X.S. Feng, M. Zhang, Y.F. Zhou, Astrophys. J. Suppl. Ser. 214 (2014) 6, http://dx.doi.org/10.1088/0067-0049/214/1/6.
[26] A. Dedner, F. Kemm, D. Kr?ner, C.D. Munz, T. Schnitzer, M. Wesenberg, J. Comput. Phys. 175 (2002) 645–673, http://dx.doi.org/10.1006/jcph.2001.6961.
[27] A. Susanto, L. Ivan, H.D. Sterck, C.P.T. Groth, J. Comput. Phys. 250 (2013) 141–164.
[28] T. Tanaka, J. Geophys. Res. 98 (1993) 17251, http://dx.doi.org/10.1029/93ja01516.
[29] P.L. Roe, J. Comput. Phys. 135 (1997) 250–258.
[30] P. Chandrashekar, Commun. Comput. Phys. 14 (2013) 1252–1286.
[31] D. Derigs, A.R. Winters, G.J. Gassner, S. Walch, J. Comput. Phys. 317 (2016) 223–256.
[32] T. Barth, VKI Computational Fluid Dynamics Volume 2 66 P (SEE N90-27993 22-34), Vol. 2, 1990.
[33] K.G. Powell, P.L. Roe, T.J. Linde, T.I. Gombosi, D.L.D. Zeeuw, J. Comput. Phys. 154 (1999) 284–309.
[34] M.S. Yalim, D. Vande. Abeele, A. Lani, T. Quintino, H. Deconinck, J. Comput. Phys. 230 (2011) 6136–6154, http://dx.doi.org/10.1016/j.jcp.2011.04.020.
[35] L. Ivan, H.D. Sterck, S.A. Northrup, C.P.T. Groth, J. Comput. Phys. 255 (2013) 205–227, http://dx.doi.org/10.1016/j.jcp.2013.08.008.
[36] V. Venkatakrishnan, AIAA Paper 93-0880, 1993.
[37] H.V. De. Ven, B.E. Niemanntuitman, A.E.P. Veldman, J. Comput. Appl. Math. 82 (1997) 423–431.
[38] N.M. Maurits, H.V. De. Ven, A.E.P. Veldman, Comput. Methods Appl. Mech. Engrg. 157 (1998) 133–150.
[39] S.-C. Chang, Y. Wu, V. Yang, X.-Y. Wang, Int. J. Comput. Fluid Dyn. 19 (2005) 359–380.
[40] T. Ogino, R.J. Walker, Geophysical Research Letters 11 (1984) 1018–1021.
[41] T. Tanaka, J. Comput. Phys. 111 (1994) 381–389.
[42] A. Nakamizo, T. Tanaka, Y. Kubo, S. Kamei, H. Shimazu, H. Shinagawa, Journal of Geophysical Research: Space Physics 114 (2009).
[43] L.P. Yang, X.S. Feng, C.Q. Xiang, Y. Liu, X. Zhao, S.T. Wu, J. Geophys. Res. (Space Phys.) 117 (2012) A08110.
[44] D. Pahud, V. Merkin, C. Arge, W. Hughes, S. McGregor, J. Atmos. Sol.-Terr. Phys. 83 (2012) 32–38.
[45] E.N. Parker, R.E. Marshak, G.W. Johnson, Phys. Today 17 (1964) 72–72.
[46] S.T. Lepri, S.K. Antiochos, P. Riley, L. Zhao, T.H. Zurbuchen, Astrophys. J. 674 (2008) 1158–1166.