Journal of Computational Physics 231 (2012) 870–903

Contents lists available at ScienceDirect

Journal of Computational Physics

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

Adaptive numerical algorithms in space weather modeling

Gábor Tóth [a,*], Bart van der Holst [a], Igor V. Sokolov [a], Darren L. De Zeeuw [a], Tamas I. Gombosi [a], Fang Fang [a], Ward B. Manchester [a], Xing Meng [a], Dalal Najib [a], Kenneth G. Powell [a], Quentin F. Stout [a], Alex Glocer [b], Ying-Juan Ma [c], Merav Opher [d]

[a] Center for Space Environment Modeling, University of Michigan, Ann Arbor, MI 48109-2143, United States
[b] Sciences and Exploration Directorate, NASA Goddard Space Flight Center, Greenbelt, MD 20771, United States
[c] Institute of Geophysics and Planetary Physics, University of California, Los Angeles, CA 90095-1560, United States
[d] Department of Astronomy, Boston University, Boston, MA 02215, United States

ARTICLE INFO

Article history:
Available online 10 March 2011

Keywords:
65D99 Numerical approximation
77A05 Magnetohydrodynamics

ABSTRACT

Space weather describes the various processes in the Sun–Earth system that present danger to human health and technology. The goal of space weather forecasting is to provide an opportunity to mitigate these negative effects. Physics-based space weather modeling is characterized by disparate temporal and spatial scales as well as by different relevant physics in different domains. A multi-physics system can be modeled by a software framework comprising several components. Each component corresponds to a physics domain, and each component is represented by one or more numerical models. The publicly available Space Weather Modeling Framework (SWMF) can execute and couple together several components distributed over a parallel machine in a flexible and efficient manner. The framework also allows resolving disparate spatial and temporal scales with independent spatial and temporal discretizations in the various models.

Several of the computationally most expensive domains of the framework are modeled by the Block-Adaptive Tree Solarwind Roe-type Upwind Scheme (BATS-R-US) code that can solve various forms of the magnetohydrodynamic (MHD) equations, including Hall, semi-relativistic, multi-species and multi-fluid MHD, anisotropic pressure, radiative transport and heat conduction. Modeling disparate scales within BATS-R-US is achieved by a block-adaptive mesh both in Cartesian and generalized coordinates. Most recently we have created a new core for BATS-R-US: the Block-Adaptive Tree Library (BATL) that provides a general toolkit for creating, load balancing and message passing in a 1, 2 or 3 dimensional block-adaptive grid. We describe the algorithms of BATL and demonstrate its efficiency and scaling properties for various problems.

BATS-R-US uses several time-integration schemes to address multiple time-scales: explicit time stepping with fixed or local time steps, partially steady-state evolution, point-implicit, semi-implicit, explicit/implicit, and fully implicit numerical schemes. Depending on the application, we find that different time stepping methods are optimal. Several of the time integration schemes exploit the block-based granularity of the grid structure.

The framework and the adaptive algorithms enable physics-based space weather modeling and even short-term forecasting.

* Corresponding author.
E-mail address: gtoth@umich.edu (G. Tóth).

0021-9991/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2011.02.006

1. Introduction

Space weather involves the physical processes in the Sun–Earth system that affect human life and technology. The most dramatic space weather events are giant eruptions, or Coronal Mass Ejections (CMEs) starting at the solar surface, expanding into the heliosphere and going by the Earth and further on at speeds around a 1000 km/s or even more. The sudden change of the solar wind speed, plasma density and interplanetary magnetic field create magnetic storms in the magnetosphere that is formed by the interaction of the solar wind and the Earth's magnetic field. The magnetic storms are responsible for spectacular aurorae as well as they can break down radio communication, degrade the accuracy of the global positioning system (GPS), and damage electronic instruments on satellites. In extreme cases the magnetic storm can induce high voltage spikes along power grids that can burn out transformers and cause black-outs in Northern America and Scandinavia. CMEs are often associated with solar energetic particles (SEPs) that propagate along the magnetic field and can reach the Earth in a matter of tens of minutes. The shock waves created by the CMEs can also accelerate particles to high energies. The increased radiation can be harmful to astronauts, the crew and passengers of flights going near the polar regions. It can also destroy sensitive onboard instruments in satellites. Being able to model, and eventually predict, space weather is important for safety as well as for protecting technology.

Physics-based space weather modeling is a challenging problem. Fig. 1 shows the various space weather domains and physical processes on a spatial scale versus temporal scale plot. In the lower left corner the auroral zone is characterized by spatial scales of about 100 km and temporal scales of a second. At the other extreme the size of the Solar system is of the order of 100 astronomical units (AU), and the solar cycle drastically changes the solar activity every 11 years. It is not feasible, or even necessary, to capture all these phenomena in a single model. But even if we model a single space weather event, a CME takes one to three days to travel 1 AU from the Sun to the Earth, while the effects in the upper atmosphere occur on a scale of seconds and kilometers. The physics of the various domains varies substantially as well. The solar convection zone is driven by radiative cooling; the solar corona, the heliosphere and the outer magnetosphere consist of low density fully ionized plasma with magnetic field; the inner magnetosphere and the radiation belt contain trapped high energy particles that are not in thermal equilibrium; while the upper atmosphere contains many different ionized as well as neutral atoms and molecules at high enough densities to make collisions important.

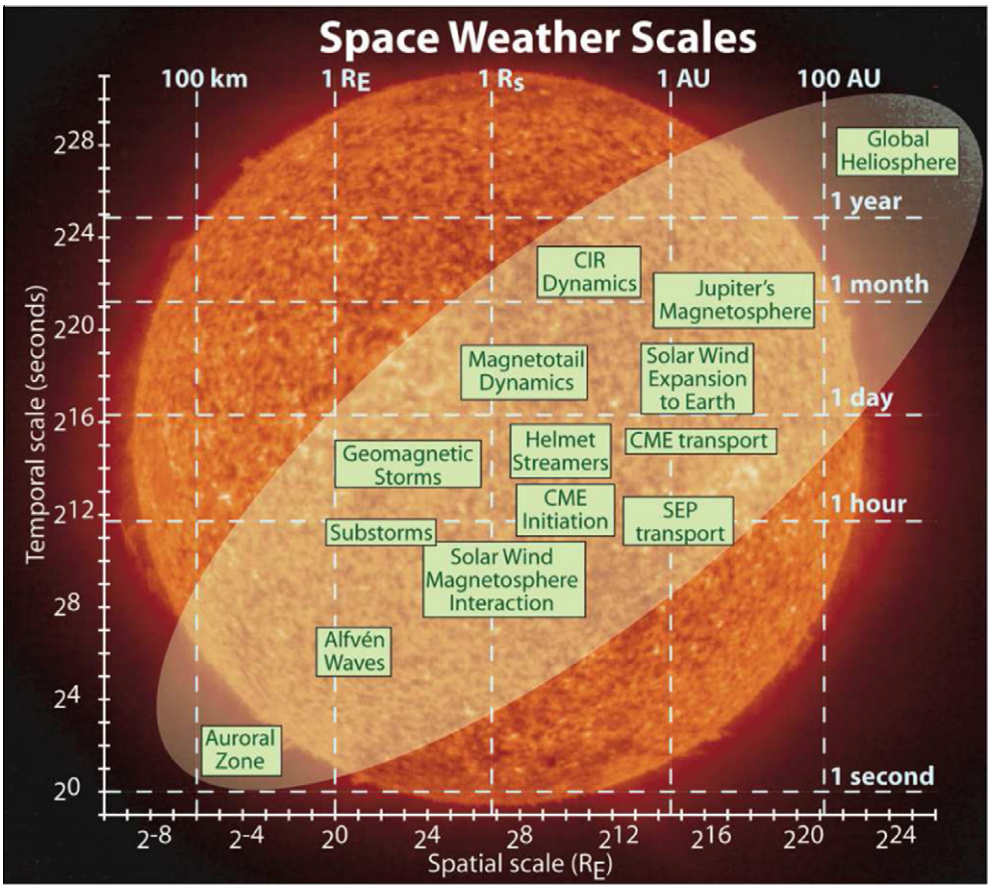

Fig. 1. Spatial and temporal scales of space weather.

Software frameworks are suitable to model multi-physics systems, because they can use different models for the different domains of the system. The domains may be adjacent to each other or overlap in space. Each model can solve different equations using different spatial and temporal discretizations. The framework is responsible for the parallel execution of the models and the data transfer required by the coupling of the models. A framework can also provide various utilities and a shared library of general algorithms, like linear solvers, interpolation algorithms, timing utilities, etc.

There are only a couple of software frameworks developed for physics-based space weather modeling. The Center for Integrated Space weather Modeling (CISM) has developed a loosely coupled framework [1] where each model runs as a separate executable. The executables are either coupled by flat files or by a general communication library Intercom [2] and the grid interpolation can be handled by the Overture library [3]. The use of these libraries has been demonstrated with a limited number of models and couplings so far [1]. The CISM framework minimizes changes to the original physics models.

The Space Weather Modeling Framework (SWMF) [4,5] has followed a different strategy. Each physics domain in the SWMF corresponds to a component. Each component is represented by one or more component versions. A component version is a physics model plus the appropriate wrappers and couplers. The components are compiled into libraries, and they are linked to the core of the framework and the shared libraries to form a single executable. The SWMF distributes the components over a parallel machine, and executes and couples them in an efficient manner [6] using the Message Passing Interface (MPI) library for communication. We note here that the physics models can also be compiled into individual executables and used as stand-alone codes. The software architecture of the SWMF is similar to the Earth System Modeling Framework (ESMF) [7] and we have demonstrated that the SWMF can indeed be coupled with an ESMF component.

While CISM's main objective was to minimize code changes, the SWMF development optimizes for ease of use, efficiency, and flexibility. We have to do substantial code changes (and development) when a new model is incorporated into the SWMF, but the resulting software is easy to use, it can be run on any parallel machine (no need to start different codes in a single parallel job), and it can run efficiently due to the flexible and efficient parallel execution algorithm [6]. Since the SWMF has many more users than developers, the benefits of our approach greatly outweigh the costs.

The SWMF provides a user-friendly interface. While the original input and output formats of the various models, currently there are ten, were quite different from each other, the models in the framework are controlled by a single input parameter file written in a general, flexible and user-friendly format that is part of the General Input Parameter Handling Toolkit (GIPHT) developed for the SWMF. GIPHT is described in some detail in Appendix A. The model outputs are also regulated and unified to some extent. This uniformity helps the users to master running the multitude of models with a less steep learning curve.

The computationally most expensive domains of the SWMF are modeled by the Block-Adaptive Tree Solarwind Roe-type Upwind Scheme (BATS-R-US) code [8,9]. In the past decade BATS-R-US has evolved from a single purpose ideal MHD model into a general, highly modular space physics modeling tool. This distinguishes BATS-R-US from many other MHD codes used in space weather modeling, which are typically applied only in a single domain. For example in the CISM framework the solar corona, the heliosphere and the magnetosphere are modeled by the MAS [10], the ENLIL [11] and the LFM [12] codes, respectively. In the SWMF all three of these domains, and several others, are modeled by a single code, BATS-R-US. This is achieved by a layered modular software architecture, where the numerical schemes are general, while the equation and application specific codes are encapsulated into interchangable modules.

BATS-R-US uses a block-adaptive mesh with either Cartesian or generalized coordinates that includes spherical, cylindrical and even toroidal grids. While block-adaptive grids are ubiquitous in astrophysics, aerospace engineering, and the general CFD community, many MHD codes in space physics, like MAS, ENLIL, LFM, and OpenGGCM [13], use static although typically stretched grids. The GUMICS code [14] uses a cell-based adaptive grid but it is restricted to first order accurate schemes and serial execution. The MPI-AMRVAC [15] code is the closest to BATS-R-US: it also uses a block-adaptive grid and it is also designed for multiple (mostly astrophysical) applications.

BATS-R-US was designed around the block-adaptive grid, and it is an integral part of the software. As we kept extending the range of applications, the limitations of the original design became apparent, for example using two-dimensional adaptive grids was not possible. To further enhance the capabilities of BATS-R-US, we have created a new Block-Adaptive Tree Library (BATL) that provides a general toolkit for creating, load balancing and message passing in a 1, 2 or 3 dimensional block-adaptive grid. Currently BATL can be used as an alternative to the original BATS-R-US core in a limited range of applications, but eventually it will completely replace the core of BATS-R-US, and it will also be available for other numerical codes. There are a number of similar libraries available, like PARAMESH [16], CHOMBO [17], SAMRAI [18], but we opted to develop BATL from scratch, so that it is fully compatible with the needs of BATS-R-US, and it requires a minimal overhead only. This paper describes the algorithms of BATL and demonstrates its efficiency and scaling properties.

Unlike most space physics MHD codes, BATS-R-US has several time-integration schemes to optimally adapt to the particular application. Local time stepping allows fast convergence towards steady state. When only a small part of the solution evolves in time, we can use a partially steady state algorithm. For fast moving waves and shocks the explicit time stepping is optimal. Stiff source terms can be handled with a point-implicit scheme. Radiative transfer and heat conduction require semi-implicit time discretization. When the whole system of equations is stiff, a fully implicit time stepping can be employed. The explicit and implicit time stepping can be combined when only part of the computational domain requires the implicit scheme [19]. Many of these algorithms operate on a block-by-block basis, thus the temporal and spatial adaptivity are interrelated.

The framework and the adaptive numerical schemes employed in the models enable us to model space weather events faster than real time with reasonable spatial and temporal resolutions. We have simulated the October 28, 2003 coronal mass ejection and the resulting magnetic storm (the most geo-effective of the so-called Halloween storms) from the solar surface to the thermosphere of the Earth faster than real time on 256 SGI Altix CPUs [20,21].

The SWMF is already used for short-term forecasting of space weather. The global magnetosphere (BATS-R-US), the ionosphere and the radiation belt models of the SWMF have been running at real time speed for several years 24/7 on a small (currently 56-core) cluster at the Community Coordinated Modeling Center (CCMC) at NASA Goddard Space Flight Center. The simulation is driven by real-time satellite data, and it provides an about 1 hour forecast (the time it takes for the solar wind to propagate from the ACE satellite location to the inflow boundary of the model) that is available as part of the Integrated Space Weather Analysis system (ISWA) [22], where it is used to predict magnetospheric configuration, radiation belt state, electric potential, and even the induced electric spikes in powerlines [23].

The SWMF source code with all its components is freely available after registration at [4]. Several of the SWMF components are accessible at the CCMC [24] for runs-on-request. This paper reviews and describes in some detail the current capabilities of the SWMF and BATS-R-US, focusing on the adaptive numerical techniques we have developed over the past decade. Section 2 describes the techniques used to adapt to the varying physics, Section 3 addresses spatial adaptation, and Section 4 describes adaptation in the time discretization. We conclude with Section 5.

2. Adaptive physics

We employ various strategies to adapt our simulation tools to the multitude of physical systems that are part of space weather. At the highest level we use the framework to divide the system into individual physics domains, and we model each domain with a different component of the framework. This allows us to use different equations, spatial and temporal discretizations in each domain. At the level of a single component, we can use a modular software architecture to allow for a variety of equations, initial and boundary conditions, source terms, and numerical schemes.

2.1. Adaptive physics in the SWMF

Fig. 2 shows the structure of the SWMF. There are about a dozen components or physics domains represented by the thumbnail pictures. The green arrows show how the domains are coupled together according to the physical processes represented by the physics models. For example the inner heliosphere is one-way coupled to the global magnetosphere model, because the solar wind speed is essentially always super-magnetosonic at the upstream boundary of the global magnetosphere model. On the other hand, most of the physics domains near the Earth are two-way coupled. In an actual simulation

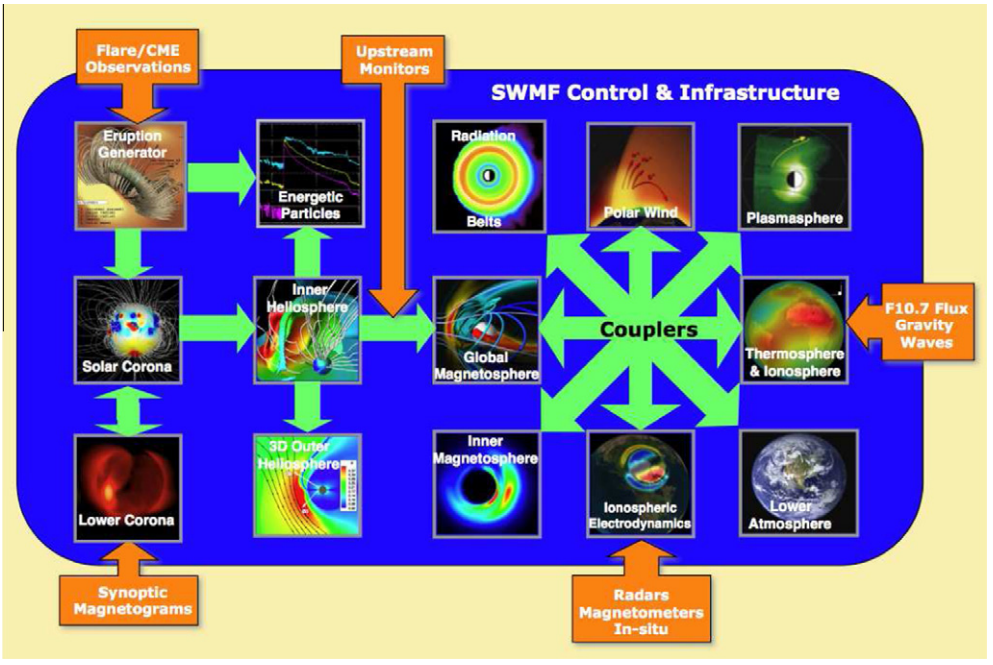

Fig. 2. Components (boxes) and their couplings (green arrows) in the Space Weather Modeling Framework. External input is indicated by the orange arrows. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

one can use any meaningful subset of the components with the selected physics models. If the simulation starts from the Sun, it is typically driven by synoptic magnetogram data, differential emission measure tomography, and flare and CME observations. Simulations restricted to magnetospheric components are usually driven by the solarwind data obtained by satellites upstream of the Earth, for example ACE, Wind or Geotail. We also use the F10.7 flux for some of the empirical relationships in the ionosphere, thermosphere, radiation belt and polar wind models.

2.1.1. Layered architecture

The SWMF has a layered architecture as shown in the left panel of Fig. 3. The top layer is an optional graphical user interface. This interface can be used to configure and compile the code, to construct and check the input parameter file, to run the code and to visualize the output. The user can also execute various scripts from the command line to do the same steps.

The second layer contains the control module, which is responsible for distributing the active components over the parallel machine, execute the models, and couple them at the specified frequency. The control module can run the models in multiple "sessions". In each session the SWMF and the models can read in a new set of input parameters and then run until the specified number of iterations or simulation time is reached. At that point a new session can start. The last session completes the whole simulation. The SWMF can also instruct the models to save restart files.

The third layer contains the physics domain components. Each component can have multiple versions. Each component version consists of a physics model with a wrapper and one or more couplers. The wrapper is an interface with the control module, while each coupler is an interface with another component. The wrapper consists of a small number of standard methods that instruct the model to initialize, set input parameters, run, finalize or save a restart file. The couplers are not standardized, because there are many very different couplings in the SWMF. On the other hand the existing couplers follow a few well defined patterns, and they can be used as templates for future couplings. The physics models can also be compiled into stand-alone executables. In this case a small main program is added to the libraries containing the physics model and the optional shared library and utilities.

The fourth and lowest layer contains the shared library and the utilities that can be used by the physics models as well as by the SWMF core. These libraries can also be used by the standalone physics models if they are compiled that way. The libraries provide physical and numerical constants, conversion between coordinate systems and time units, interpolation on regular and irregular grids, sorting, linear solvers with preconditioners, advection schemes, various modules for the parallel ray tracing algorithms, reading input parameter files, saving and reading plot files and lookup tables. The utilities contain various algorithms to read and process different space physics data files, empirical models for various space physics systems, a library for timing and profiling, and a NOMPI library that allows compiling the code in a serial mode when the MPI library is not needed/available.

2.1.2. Physics-based and empirical models

Table 1 shows the current components of the SWMF and the physics-based and empirical models that can represent these components. In practice the SWMF is almost never run with all its components at the same time, but we typically use a

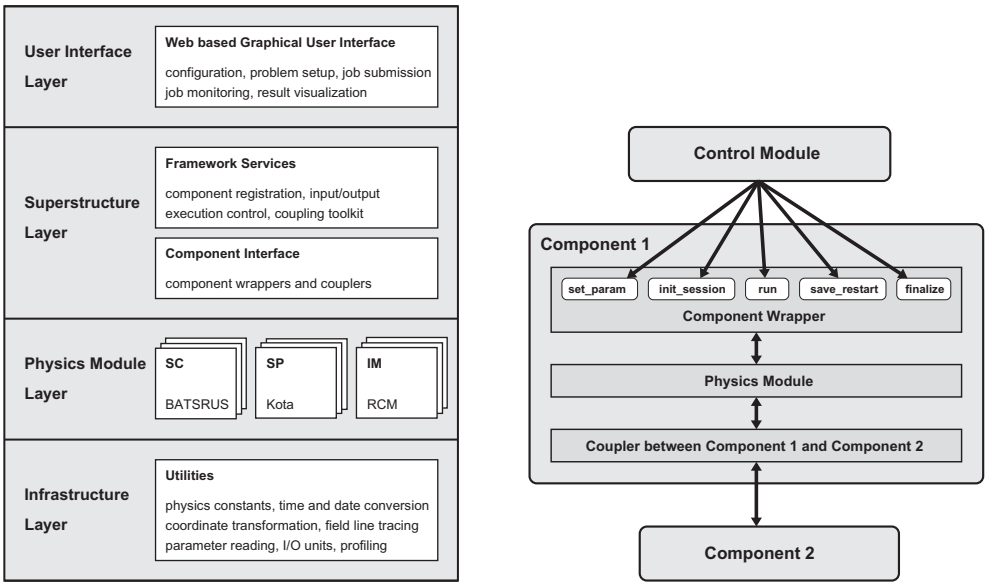

Fig. 3. The layered architecture of the SWMF (left) and the structure of the components (right). See Table 1 for the abbreviations used for the physics components.

Table 1
Physics-based and empirical models of the SWMF.

	Component name	ID	Physics-based/ empirical models
1	Eruptive event generator	EE	BATS-R-US/breakout, flux-rope
2	Lower corona	LC	BATS-R-US
3	Solar corona	SC	BATS-R-US
4	Inner heliosphere	IH	BATS-R-US
5	Outer heliosphere	OH	BATS-R-US
6	Solar energetic particles	SP	Kota, FLAMPA
7	Global magnetosphere	GM	BATS-R-US/Tsyganenko
8	Inner magnetosphere	IM	RCM, CRCM, HEIDI, RAM-SCB
9	Radiation belt	RB	RBE
10	Polar wind	PW	PWOM
11	Ionosphere electrodynamics	IE	RIM/ Weimer
12	Upper atmosphere	UA	GITM/MSIS, IRI

subset of the available components. For example we can run together the solar corona (SC) and the inner heliosphere (IH) models driven by solar synoptic magnetograms, or the global magnetosphere (GM), inner magnetosphere (IM) and ionosphere electrodynamics (IE) models driven by satellite observations of the solar wind. In some cases the physics-based models can be replaced by an empirical model, for example an inner magnetosphere model maybe run in standalone mode using the empirical Tsyganenko [25] and Weimer [26,27] models to represent the magnetic field of the magnetosphere and the electric potential of the ionosphere, respectively.

The empirical models use various observations as input parameters. These models are rather inexpensive to run, so they can be executed as simple subroutine and function calls. On the other hand the empirical models typically provide only partial information about the physical state of a domain, for example the Tsyganenko model provides the magnetic field but not the other plasma parameters. This means that the empirical and physics-based models are fundamentally different, and their implementation into the SWMF reflects this. The empirical models are simple libraries in the utility layer of the architecture which can be accessed by any of the physics models both from the SWMF and in stand-alone mode. On the other hand the physics models are implemented in the component layer with the wrappers and couplers as shown in Fig. 3.

As Table 1 shows, several components can be represented by the BATS-R-US code. Since the SWMF is compiled into a single library, the components cannot contain modules, external subroutines or functions with identical names. We have an automated script that copies the BATS-R-US code into separate directories and replaces all the potentially conflicting names with a unique name, for example `module ModFaceFlux` is renamed to `IH_ModFaceFlux` in the IH component. This way the renamed BATS-R-US codes representing various components can be compiled together and they can be configured and run with different parameters. Next we briefly describe each component and the corresponding model(s).

The *Eruptive Event* (EE) generator component is responsible for creating a CME. This may be done by a physics-based model of flux emergence from the convection zone, or by much simpler, and less expensive empirical models that insert an unstable flux rope into the steady solar corona solution, or insert an arcade and apply shearing motion at the lower boundary of the corona model [28,29]. In the physics-based EE model, BATS-R-US solves the MHD equations with an optically thin radiative loss and empirical coronal heating term in a box of several megameters that should ideally include a whole active region and simulate the emergence of a flux rope from the convection zone. Currently the physics-based EE model only works in a stand-alone mode [30,31], and we use the empirical models to generate CMEs in the SWMF [32,33].

The *Lower Corona* (LC) domain starts at the photosphere and goes out to a few solar radii into the solar corona. BATS-R-US solves the MHD equations with empirical heating functions, heat conduction, and radiative cooling on a co-rotating spherical grid with highly stretched radial coordinates to capture the transition region [34].

The *Solar Corona* (SC) model describes the corona out to about 25 solar radii. In our latest model, BATS-R-US solves the two-temperature MHD equations with Alfvén wave heating and heat conduction on either Cartesian or spherical grid in a frame corotating with the Sun [35].

The *Inner Heliosphere* (IH) model typically extends from about 20 solar radii to the orbit of the Earth and has been extended to 10 AU. BATS-R-US solves the ideal or two-temperature MHD equations on a Cartesian grid in either co-rotating or inertial frame, and it can model the propagation of CMEs from the Sun to the Earth [36,37,5].

The *Outer Heliosphere* (OH) extends from 30 AU to1000 AU, beyond the outer edges of the solar system. BATS-R-US solves the MHD equations for ions combined with the hydrodynamic equations for multiple neutral populations [38,39].

The *Solar Energetic particle* (SP) domain consists of one or more one dimensional field lines, which are assumed to advect with the plasma. The solar energetic particles accelerate and diffuse along the field lines using either the FLAMPA [40] or the Kota SEP [41] models.

The *Global Magnetosphere* (GM) domain surrounds the Earth and it extends about 30 Earth radii (R_E) towards the Sun, a few hundred R_E towards the magnetotail, and about 60 R_E in the orthogonal directions. BATS-R-US solves the ideal, semi-relativistic, Hall, anisotropic, or multi-ion MHD equations on a Cartesian or possibly spherical grid [42–44]. The Tsyganenko empirical models can provide the magnetic field as a function of observed solar wind parameters and planetary indexes [25].

The *Inner Magnetosphere* (IM) consists of the closed magnetic field line region around the Earth. The RCM [45,46] solves for the bounce averaged and isotropic but energy resolved particle distribution of electrons and various ions. The CRCM [47], HEIDI [48] and RAM-SCB [49,50] models also resolve the pitch angle distribution.

The *Radiation Belt* (RB) domain coincides with IM but it models the relativistic electrons. The RBE [51,52] model solves the bounce-averaged Boltzmann equation.

The *Polar Wind* (PW) domain consists of the open magnetic field line region near the Earth. The PWOM [53] solves the field-aligned hydrodynamic equation for electrons and several ions along many field lines. The field lines are advected by the ionospheric drift.

The *Ionospheric Electrodynamics* (IE) model is a two dimensional height-integrated spherical surface at a nominal ionospheric altitude (at around 110 km for the Earth). The RIM [54] code uses the field-aligned currents to calculate particle precipitation and conductances based on empirical relationships, and then it solves for the electric potential on a 2D spherical grid. There are also several empirical models for IE, including Weimer's models [26,27].

The *Upper Atmosphere* (UA) contains the thermosphere and the ionosphere extending from around 90 km to about 600 km altitude for the Earth. The GITM [55] code solves the equations of multi-species hydrodynamics with viscosity, thermal conduction, chemical reactions, ion-neutral friction, source terms due to solar radiation, etc. on a spherical grid in a corotating frame. The MSIS [56] and IRI [57] empirical models provide statistical average states for the upper atmosphere and ionosphere, respectively. These can be used to define the lower boundary conditions for the Polar Wind model, for example.

2.1.3. Model coupling

The various models are coupled together at regular intervals, either based on simulation time or iteration number. The relevant physical quantities are passed with efficient MPI communication. Deadlocks are carefully avoided [6]. In addition to transferring the data, we have to transform between coordinate systems, take care of unit conversions, and interpolate between different grids. Often the models are moving or rotating relative to each other so that the mapping has to be recalculated every coupling time. A further complication arises for adaptive grids that may change between two couplings. We have developed utilities to take care of coordinate transformations and interpolation between various grids. Unit conversions are handled by requiring that all information passed between models are in SI units.

Since the models use widely different grids and time steps, coupling through a simple interface may not work well, especially when the flow is slower than the fast magnetosonic speed. A possible solution is to overlap the models. For example the inner boundary of the inner heliosphere model is provided by the solar corona model at 20 solar radii, while SC obtains its outer boundary conditions from IH at 24 solar radii. The overlap serves as a buffer to suppress numerical artifacts due to the differences between the spatial and temporal resolutions.

In some cases the coupling between the physics models requires some complicated and expensive calculations. For example the inner magnetosphere and the radiation belt models require the magnetic field geometry and the plasma state along the closed magnetic field lines of the global magnetosphere model. Since the GM grid is large and it is distributed over many processors, the tracing of magnetic field lines is quite challenging. We have developed a couple of highly parallel and efficient schemes for tracing multiple field lines [42,52] that allows us to obtain mapping information, integrate quantities along the lines, or extract state variables and positions along the lines.

2.2. Adaptive physics in BATS-R-US

BATS-R-US plays many roles in the SWMF: it models the EE, LC, SC, IH, OH and GM components. In each of these models, and in many other applications, BATS-R-US solves different sets of equations: radiative, ideal, Hall, two-fluid, anisotropic, semi-relativistic, multi-species or multi-fluid MHD. In addition to the basic equations, there are various source terms that also change from application to application: collisions, charge exchange, chemistry, photo-ionization, recombination, etc. The boundary and initial conditions vary greatly as well.

We introduced a layered modular software architecture, as shown in Fig. 4, to handle all these applications with a single base code. The state variables of the equation system are defined by the equation modules, while the rest of the application dependent details are implemented into user modules. The `Config.pl` script is used to select the equation and user modules that are compiled together with the code. There are currently 37 equation modules and 42 user modules (obviously not all combinations are possible) which means that BATS-R-US can be configured for quite a few different applications. The equation modules are about 150 lines only, the user modules vary from a few hundred to a few thousand lines, depending on the number and complexity of the various methods implemented. For each user method there is an 'empty' version that is compiled by default. This allows us to add new user methods without modifying the existing user modules.

2.3. Systems of equations in BATS-R-US

In this section, we describe the basic sets of the equations that are implemented in BATS-R-US: hydrodynamics, ideal MHD, Hall-MHD with electron physics, anisotropic MHD, MHD with Alfvén wave heating and electron thermal heat conduction, multi-species and multi-fluid MHD. All these governing equations are recast as

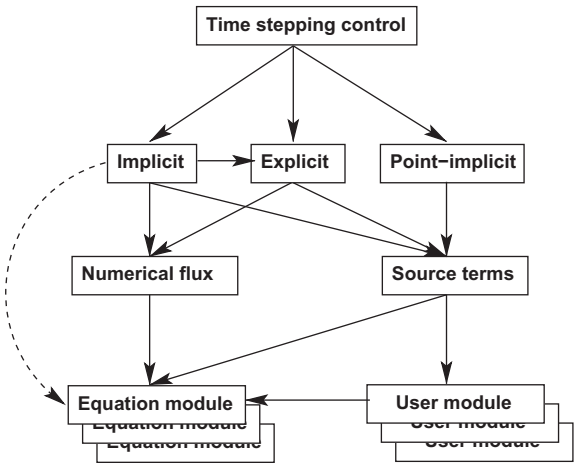

Fig. 4. The layered software structure of BATS-R-US. The arrows point from the module that is using data or methods from the other module. There are multiple versions of the equation and user modules containing the equation variable definitions and the application specific code, respectively. The numerical fluxes depend on the selected equation module, the source terms may also be defined in the user module. The various time stepping schemes are independent of the details of the equations being solved with the possible exception of the implicit preconditioner.

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{U}) = \mathbf{S}, \tag{1}$$

where **U** denotes the vector of conserved state quantities, **F** is the flux vector, and **S** indicates source terms that cannot be written as a divergence of a flux. These include external sources like gravity, Coriolis and centrifugal forces, charge exchange with neutrals, radiative cooling, photoionization, as well as internal sources, like chemical reactions or collisions among different ion species. Many of these terms are application dependent, and they are given in the user modules. There are also several 'source' terms that contain spatial derivatives, but cannot be written in a pure divergence form.

2.3.1. Hydrodynamics

The simplest set of equations in BATS-R-US are the Euler equations for hydrodynamics:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0, \tag{2}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u} + Ip) = 0, \tag{3}$$

$$\frac{\partial \varepsilon}{\partial t} + \nabla \cdot (\mathbf{u}\varepsilon + \mathbf{u}p) = 0, \tag{4}$$

where I is the identity matrix and the primitive variables ρ, **u**, and p are the mass density, velocity, and the thermal pressure, respectively. The total hydrodynamic energy density is

$$\varepsilon = \frac{p}{\gamma - 1} + \frac{\rho u^2}{2}, \tag{5}$$

where γ is the adiabatic index. The hydrodynamics equations are used for code verification with standard test problems. The equations also apply to the neutral fluids in multifluid MHD.

2.3.2. Ideal magnetohydrodynamics

The ideal MHD equations can be written in (near) conservative form as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0, \tag{6}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot \left[\rho \mathbf{u}\mathbf{u} + I\left(p + \frac{1}{2}B^2\right) - \mathbf{B}\mathbf{B}\right] = -\mathbf{B}\nabla \cdot \mathbf{B}, \tag{7}$$

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{B} - \mathbf{B}\mathbf{u}) = -\mathbf{u}\nabla \cdot \mathbf{B}, \tag{8}$$

$$\frac{\partial e}{\partial t} + \nabla \cdot \left[\mathbf{u}\left(e + p + \frac{1}{2}B^2\right) - \mathbf{u} \cdot \mathbf{B}\mathbf{B}\right] = -\mathbf{u} \cdot \mathbf{B}\nabla \cdot \mathbf{B}, \tag{9}$$

where **B** is the magnetic field in normalized units so that the magnetic permeability of vacuum $\mu_0 = 1$. The total energy density is

$$e = \frac{p}{\gamma - 1} + \frac{\rho u^2}{2} + \frac{B^2}{2}, \tag{10}$$

The source terms proportional to $\nabla \cdot \mathbf{B}$ on the right hand sides were introduced to control the numerical error in $\nabla \cdot \mathbf{B}$ [58,8]. In this 8-wave formulation the numerical $\nabla \cdot \mathbf{B}$ errors are propagated together with the flow, leading to improved robustness and accuracy of the numerical scheme.

There are several alternative methods implemented in BATS-R-US to maintain the solenoidal constraint. The projection [59] scheme can remove the $\nabla \cdot \mathbf{B}$ error by solving an elliptic problem. The constrained transport scheme [60] generalized to AMR grids [61] maintains the $\nabla \cdot \mathbf{B} = 0$ condition to round-off error at the expense of staggering the magnetic field. For the MHD Roe solver, we have also developed a new 7-wave scheme [62]. Finally, in the hyperbolic/parabolic cleaning method of [63], a new scalar field is added to the MHD variables that couples the induction equation to the $\nabla \cdot \mathbf{B}$ error:

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{B} - \mathbf{B}\mathbf{u}) = -\nabla \varphi, \tag{11}$$

$$\frac{\partial \varphi}{\partial t} + c_h^2 \nabla \cdot \mathbf{B} = -\frac{c_h^2}{c_p^2}\varphi, \tag{12}$$

Solving the modified induction equation and the equation for φ will make the $\nabla \cdot \mathbf{B}$ error propagate with c_h and diffuse by c_p. Both c_h and c_p has to be uniform over the computational domain, but they can vary with time. We typically set c_h to be somewhat less than the largest wave speed in the domain, so that the time step is not limited by the hyperbolic cleaning. The parabolic term $-(c_h^2/c_p^2)\varphi$ is evaluated in a split and implicit way. After updating φ^n at time level n with the pure hyperbolic term, the obtained φ^* value is further modified to

$$\varphi^{n+1} = \frac{\varphi^*}{1 + \Delta t c_h^2/c_p^2}, \tag{13}$$

where Δt is the time step. This scheme is unconditionally stable, so the value of c_p does not limit the explicit time step.

In regions of small thermal to magnetic and/or dynamic pressure the conservative scheme can occasionally produce negative pressure. We therefore also have the option to solve for the pressure

$$\frac{\partial p}{\partial t} + \nabla \cdot (p\mathbf{u}) = -(\gamma - 1) p \nabla \cdot \mathbf{u}, \tag{14}$$

instead of the energy equation. The pressure equation should not be used in regions where shocks occur, because the non-conservative equation will not recover the correct jump conditions. An additional problem arises when the pressure equation without explicit resistivity is applied in a region where the magnetic field reconnects (due to numerical diffusion). In this case the total energy is not conserved, and some of the magnetic energy is lost without producing the corresponding Joule heating. Therefore we are solving both the pressure and energy equations, and switch between them according to some criteria, for example shocks can be identified by steep pressure gradients. We have also implemented geometric criteria, like the inside of a sphere around a planet, to define regions where no shocks or reconnection are expected.

It is often advantageous to split the magnetic field into an analytic part and a numerical part $\mathbf{B} = \mathbf{B}_0 + \mathbf{B}_1$ [64], where the analytic $\mathbf{B}_0$ is divergence free, but not necessarily curl free. We allow $\mathbf{B}_0$ to be time dependent. The advantage of this splitting is the improved accuracy and robustness of the MHD scheme in regions where gradients of $\mathbf{B}_0$ are large. Typically $\mathbf{B}_0$ captures the dipole field of a planet or the magnetic field of an active region. The rotation of a tilted planetary dipole field results in a time dependent $\mathbf{B}_0$ field. In solar applications B_0 is calculated as a potential field up to some radius (the "source surface") and it is taken to be radial outside. If the radius of the source surface is inside the computational domain, as is usually the case, there will be a finite current $\nabla \times \mathbf{B}_0$ there. The source terms containing the $\partial \mathbf{B}_0/\partial t$ and $\nabla \times \mathbf{B}0$ terms in the momentum, induction and energy equations are shown in [65].

Fig. 5 shows a simulation of a magnetic flux rope emerging from the convection zone into the corona through the photosphere [30]. This is a physics-based mechanism to produce coronal mass ejections. The MHD equations are solved with a non-ideal equation-of-state based on the OPAL tables [66]. There are also source terms corresponding to the 8-wave scheme, as well as gravity, radiative cooling, and an empirical heating function. The computational domain is a box with $0 < x,y < 10{,}000$ km and $-10{,}000 < z < 5000$ km in the vertical direction. The photosphere is at $z = 0$. The grid uses four levels of grid refinement with 25 million cells ranging from 35 to 560 km in size.

2.3.3. Semi-relativistic MHD

The classical Alfvén speed $B/\sqrt{\rho}$ is not bound by the speed of light c. Near strongly magnetized planets $B/\sqrt{\rho}$ can indeed exceed c. In these applications we use the semi-relativistic MHD equations [67,65] that retain the displacement current $\partial \mathbf{E}/\partial t$ in the induction equation and provide the physically correct limit on the wave speeds. This is also helpful in allowing larger time steps and reducing the numerical diffusion which is proportional to the wave speeds in the shock capturing schemes. In magnetospheric simulations it is customary [12,13] to artificially reduce the speed of light to $c' < c$ so that the simulation runs faster and the numerical diffusion is smaller. While this so-called Boris correction [67] is not fully justified in time accurate simulations, it often results in more accurate results at a lower cost than solving the ideal MHD equations or the

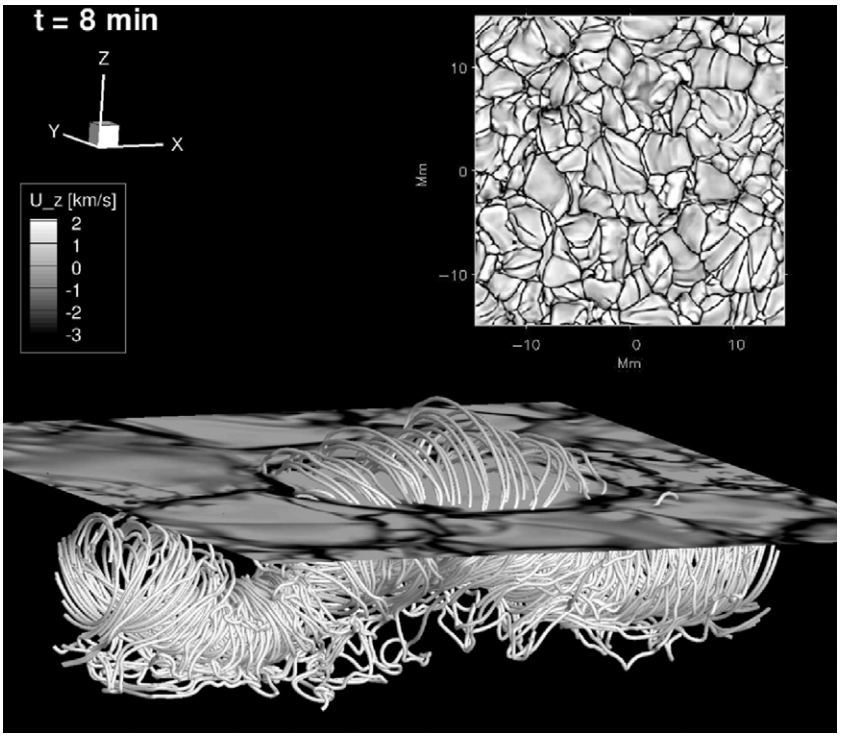

Fig. 5. Simulation of a magnetic flux rope (white tubes) emerging from the convection zone into the corona. The surface of the photosphere is shown by the horizontal plane that is colored according to the vertical velocity. The inset shows the same plane from above. The convection patterns look quite realistic.

semi-relativistic MHD equations with the true speed of light. See [65] for the semi-relativistic MHD equations and their various approximations, and [68] for a detailed discussion on the numerical effects of the Boris correction in magnetospheric simulations.

2.3.4. Hall magnetohydrodynamics

In ideal MHD, the electric field is simply approximated as $\mathbf{E} = -\mathbf{u} \times \mathbf{B}$. In Hall MHD, it is given by the generalized Ohm's law as

$$\mathbf{E} = -\mathbf{u} \times \mathbf{B} + \eta \mathbf{J} + \frac{\mathbf{J} \times \mathbf{B}}{q_e n_e} - \frac{\nabla p_e}{q_e n_e}. \tag{15}$$

Here, we introduced the resistivity η, the current density $\mathbf{J} = \nabla \times \mathbf{B}$, the electron pressure p_e, the electron number density n_e and the elementary charge $q_e \approx 1.6 \times 10^{-19}$ coulombs. The second term on the right hand side is the Ohmic dissipation term, the third is the Hall term and the last term is the Biermann battery term (often called the ambipolar term). The electron inertia term is neglected.

Substituting this electric field into the ion momentum and induction equations and adding a separate equation for the electron pressure, the Hall MHD equations become

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0, \tag{16}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot \left[\rho \mathbf{u}\mathbf{u} + I\left(p + p_e + \frac{1}{2} B^2 \right) - \mathbf{B}\mathbf{B} \right] = 0, \tag{17}$$

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \times \left[-\mathbf{u}_e \times \mathbf{B} + \eta \mathbf{J} - \frac{\nabla p_e}{q_e n_e} \right] = 0, \tag{18}$$

$$\frac{\partial p}{\partial t} + \nabla \cdot (p \mathbf{u}) = (\gamma - 1) \left[-p \nabla \cdot \mathbf{u} + \frac{2}{\tau_{ie}} (p_e - p) \right], \tag{19}$$

$$\frac{\partial p_e}{\partial t} + \nabla \cdot (p_e \mathbf{u}_e) = (\gamma - 1) \left[-p_e \nabla \cdot \mathbf{u}_e + \eta J^2 + \frac{2}{\tau_{ie}} (p - p_e) \right], \tag{20}$$

where the electron velocity vector is obtained as

$$\mathbf{u}_e = \mathbf{u} - \frac{\mathbf{J}}{q_e n_e}, \tag{21}$$

Here we assumed that the ions are singly charged so that $n_e = n_i = \rho/M_i$ where n_i is the ion number density and M_i is the ion mass. The second term on the right hand side of the electron pressure Eq. (20) is the Ohmic dissipation. The last terms in the ion and electron pressure Eqs. (19) and (20), are due to the collisional heat transfer between the electrons and ions, established on the time scale

$$\tau_{ie} = \frac{2}{3}\frac{M_i}{\eta q_e^2 n_e}. \tag{22}$$

We also solve for the total energy density:

$$\frac{\partial e}{\partial t} + \nabla \cdot \left[(\varepsilon + p)\mathbf{u} + (\varepsilon_e + p_e)\mathbf{u}_e + B^2\mathbf{u}_e - \mathbf{B}\mathbf{B}\cdot\mathbf{u}_e - \mathbf{B}\times\left(\eta\mathbf{J} - \frac{\nabla p_e}{q_e n_e}\right)\right] = 0, \tag{23}$$

where

$$e = \varepsilon + \varepsilon_e + \frac{B^2}{2}, \quad \varepsilon = \frac{p}{\gamma - 1} + \frac{\rho u^2}{2}, \quad \varepsilon_e = \frac{p_e}{\gamma - 1} \tag{24}$$

are the total energy density, the hydrodynamic ion energy density, and the internal electron energy density, respectively. Similar to the ideal MHD equations, we keep the option to switch between energy and pressure as described in Section 2.3.2.

There are two numerical challenges in the Hall-MHD equations:

1. The current and electron pressure gradient in the induction and energy equations introduce second order spatial derivatives.
2. The Hall term results in the whistler waves with a speed that is approximately inversely proportional to the wave length and therefore it is inversely proportional to the grid resolution Δx.

The first problem is overcome by using central differencing for the calculation of the current and electron pressure gradient that appears in the second order derivatives and third order interpolation is used near the resolution changes to fill in the ghost cells, so that a second order accurate discretization can be achieved [43]. The second problem can render the finite volume schemes first order, and in [43] the use of symmetric limiters, like the monotonized central limiter, is advocated to make the numerical scheme second order accurate.

An interesting test of the Biermann battery term, $-\nabla p_e/(q_e n_e)$, in the generalized Ohm's law (15) is the generation of magnetic field from zero initial magnetic field. For this test, we use the initial conditions: $\mathbf{B} = 0$, $\mathbf{u} = 0$, $n_e = n_0 + n_1\cos(k_x x)$, and $p_e = p_0 + p_1\cos(k_y y)$. The resulting magnetic field is

$$\frac{\partial B_z}{\partial t} = -\frac{k_x k_y n_1 p_1 \sin(k_x x)\sin(k_y y)}{[n_0 + n_1\cos(k_x x)]^2}. \tag{25}$$

We use $k_x = k_y = \pi/10$, $n_0 = p_0 = 1$ and $n_1 = p_1 = 0.1$ on a $|x|, |y| \leqslant 10$ double periodic domain. The solution and the error after a single time step are shown in Fig. 6. The small error is concentrated around $x = 0$ where the density n_e has a local maximum, which is clipped by the slope limiters. We have verified that the code reproduces the analytic solution with second order accuracy. Verification tests for the Hall term implementation are shown in [43].

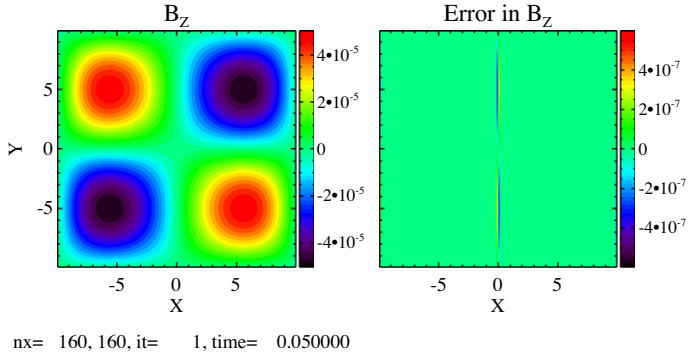

Fig. 6. Generating magnetic field with the Biermann battery term. The left panel shows B_z after a single time step at time = 0.05. The error relative to the analytical solution is shown in the right panel.

2.3.5. Pressure anisotropy

Up to this point we assumed that the ion pressure is isotropic and can be described by a single scalar quantity. In collisionless space plasmas the ion pressure distribution can become anisotropic. The ion pressure tensor can be approximated as

$$\mathbf{P} = p_\perp I + (p_\parallel - p_\perp)\mathbf{bb}, \tag{26}$$

where $p_\perp$ and $p_\parallel$ are the perpendicular and parallel pressure components with respect to the direction of the magnetic field given by $\mathbf{b} = \mathbf{B}/B$. In the current implementation, we assume that the electron pressure is isotropic, although we may generalize for anisotropic electron pressure in the future. The ion momentum equation with anisotropic pressure can be written as

$$\rho\frac{\partial \mathbf{u}}{\partial t} + \rho\mathbf{u}\cdot\nabla\mathbf{u} + \nabla\cdot\mathbf{P} = q_e n_e(\mathbf{E} + \mathbf{u}\times\mathbf{B}) - \eta q_e n_e \mathbf{J}, \tag{27}$$

where the last term on the right hand side is the collisional momentum transfer from the electrons to the ions. For the generalized Ohm's law (15), the ion momentum equation can be recast into conservation form

$$\frac{\partial \rho\mathbf{u}}{\partial t} + \nabla\cdot\left(\rho\mathbf{u}\mathbf{u} + \mathbf{P} + Ip_e + I\frac{B^2}{2} - \mathbf{B}\mathbf{B}\right) = 0. \tag{28}$$

The time evolution of the electron pressure and magnetic field is still given by (20) and (18), respectively. The perpendicular and parallel ion pressures are

$$\frac{\partial p_\perp}{\partial t} + \nabla\cdot(p_\perp\mathbf{u}) = -p_\perp\nabla\cdot\mathbf{u} + p_\perp\mathbf{b}\cdot(\nabla\mathbf{u})\cdot\mathbf{b} + (\gamma-1)\frac{2}{\tau_{ie}}(p_e - p_\perp), \tag{29}$$

$$\frac{\partial p_\parallel}{\partial t} + \nabla\cdot(p_\parallel\mathbf{u}) = -2p_\parallel\mathbf{b}\cdot(\nabla\mathbf{u})\cdot\mathbf{b} + (\gamma-1)\frac{2}{\tau_{ie}}(p_e - p_\parallel), \tag{30}$$

where we have assumed the adiabatic index to be $\gamma = 5/3$. The collisional heat transfer between the ions with anisotropic pressures and the electrons is on the electron–ion equilibration time scale τ_{ie} given in (22).

For the sake of simpler implementation in BATS-R-US, we solve for the parallel pressure and the "scalar" pressure

$$p = \frac{2p_\perp + p_\parallel}{3}, \tag{31}$$

which is a third of the trace of the pressure tensor. The advantage of this approach is that the relationship between pressure p and energy density e remains the same as in (24). When needed, the perpendicular pressure can be expressed as $p_\perp = (3p - p_\parallel)/2$.

The equations for p and e are

$$\frac{\partial p}{\partial t} + \nabla\cdot(p\mathbf{u}) = -\left(p - \frac{p_\parallel}{3}\right)\nabla\cdot\mathbf{u} + (p - p_\parallel)\mathbf{b}\cdot(\nabla\mathbf{u})\cdot\mathbf{b} + (\gamma-1)\frac{2}{\tau_{ie}}(p_e - p), \tag{32}$$

$$\frac{\partial e}{\partial t} + \nabla\cdot\left[\varepsilon\mathbf{u} + \mathbf{P}\cdot\mathbf{u} + (\varepsilon_e + p_e)\mathbf{u}_e + B^2\mathbf{u}_e - \mathbf{B}\mathbf{B}\cdot\mathbf{u}_e - \mathbf{B}\times\left(\eta\mathbf{J} - \frac{\nabla p_e}{q_e n_e}\right)\right] = 0. \tag{33}$$

The electron velocity $\mathbf{u}_e$ is defined by (21) for Hall MHD, or $\mathbf{u}_e = \mathbf{u}$ can be taken in ideal MHD. Again we have the option in BATS-R-US to switch between the energy and pressure formulations as outlined in Section 2.3.2.

As a verification example, we simulate a circularly polarized Alfvén wave propagating at the Alfvén speed

$$u_A = \sqrt{\frac{B^2 + p_\perp - p_\parallel}{\rho}}. \tag{34}$$

This wave can destabilize if the parallel pressure is large enough. Here, we will restrict ourselves to the stable wave solutions. For this test we switched off the Hall and Biermann battery terms. The initial condition consists of a uniform background $B_x = 10$, $\rho = 1$, $p_\parallel = 6$, $p_\perp = 50$, and zero velocities on the 1D periodic domain $|x| \leqslant 6$. This corresponds to the Alfvén speed $u_A = 12$. This background is modulated with a sine perturbation in u_y and u_z with amplitude 0.12 and in B_y and B_z with amplitude 0.1. The phase difference between the y and z perturbations is $\pi/2$ and the wavelength is 6. A convergence study is shown in Fig. 7 for the Rusanov scheme with the monotonized central (MC) [69] and Koren [70] slope limiters. The convergence rate is approximately second order toward the analytic solution.

Note that while the total energy Eq. (33) is in conservation form, the parallel pressure Eq. (30) cannot be recast in pure divergence form. Shock capturing schemes require conservation laws to get proper jump conditions. Energy conservation only replaces one of the two pressure equations. However, the anisotropy behind a shock is constrained by the fire-hose, mirror, and proton cyclotron instabilities. If the criteria for these instabilities are met, we reduce the anisotropy so that the plasma becomes stable again. We use the energy equation and the instability criteria to get proper jump conditions. In addition, we have implemented a relaxation term that artificially pushes the pressure towards isotropy on a given time scale τ. This relaxation can mimic the ion–ion, ion–electron, or wave–ion interactions. More detail on the implementation of the anisotropic pressure and the applications in space science will be reported elsewhere.

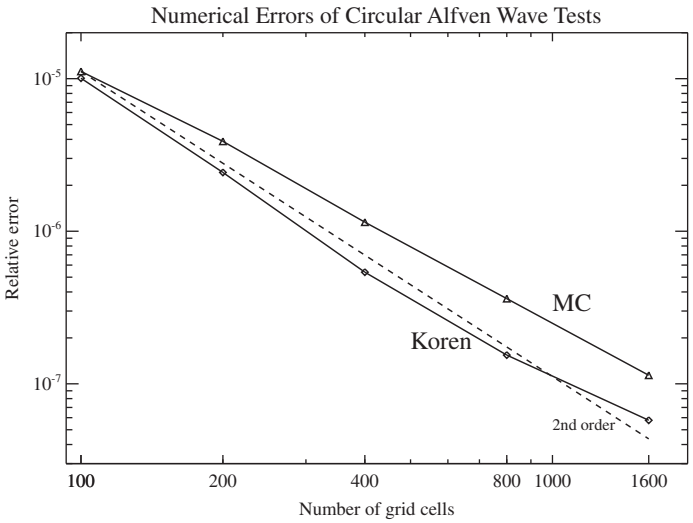

Fig. 7. Convergence study for the circular polarized Alfvén waves with anisotropic pressure. The two curves are obtained with the monotonized central (MC) and Koren limiters, respectively. The dashed line indicates the second order convergence rate.

2.3.6. Alfvén waves and heat conduction

In regimes where the space plasma is collisional, the field-aligned electron thermal heat conduction can not always be ignored. We therefore have implemented Spitzer's collisional formulation for the electron heat flux

$$\mathbf{q}_e = -\kappa_e T_e^{5/2} \mathbf{b}\mathbf{b} \cdot \nabla T_e, \tag{35}$$

where T_e is the electron temperature. The coefficient κ_e depends on the choice of the Coulomb logarithm $\ln \Lambda$. Sufficiently far from the Sun and planets, the plasma is no longer collisional so that the Spitzer formulation no longer holds. We therefore smoothly diminish the heat conduction coefficient from the full value within the given radius $r_{\text{collisional}}$ to zero beyond the given radius $r_{\text{collisionless}}$. The electron heat flux enters the electron pressure equation

$$\frac{\partial p_e}{\partial t} + \nabla \cdot (p_e \mathbf{u}_e) + (\gamma - 1)\nabla \cdot \mathbf{q}_e = (\gamma - 1)\left[-p_e \nabla \cdot \mathbf{u}_e + \eta J^2 + \frac{2}{\tau_{ie}}(p - p_e)\right]. \tag{36}$$

In the energy equations of ideal MHD, Hall-MHD, and MHD with anisotropic pressure, Eqs. (9), (23) and (33), respectively, the contribution $\nabla \cdot \mathbf{q}_e$ is added to the left hand side. For ideal MHD, we add $(\gamma - 1)\nabla \cdot \mathbf{q}_e$ to the left hand side of the pressure Eq. (14) as well. Similar to the Hall and Biermann battery terms, the heat conduction also introduces second order spatial derivatives. We use central differencing for the electron temperature gradient calculation and third order interpolation is used for T_e near the resolution changes analogous to [43].

One of the suggested mechanisms to heat and accelerate the solar wind is by the Alfvén waves. We use the Wentzel–Kramers–Brillouin (WKB) [71,72] approximation for the short wavelength Alfvén waves. The time evolution of the wave energy density is given by

$$\frac{\partial E_w^+}{\partial t} + \nabla \cdot \left[E_w^+(\mathbf{u} + \mathbf{u}_A)\right] = -(\gamma_w - 1)E_w^+ \nabla \cdot \mathbf{u} - Q^+, \tag{37}$$

$$\frac{\partial E_w^-}{\partial t} + \nabla \cdot \left[E_w^-(\mathbf{u} - \mathbf{u}_A)\right] = -(\gamma_w - 1)E_w^- \nabla \cdot \mathbf{u} - Q^-, \tag{38}$$

where $\mathbf{u}_A = \mathbf{B}/\sqrt{\rho}$ is the Alfvén speed and $\gamma_w = 3/2$ is the effective polytropic index for the Alfvén wave energy density. The '+' superscript indicates the obliquely propagating Alfvén waves in the direction of $\mathbf{B}$, and similarly the superscript '−' indicates the Alfvén waves antiparallel to $\mathbf{B}$. For the wave dissipation Q, a phenomenological description of the Kolmogorov dissipation is used

$$Q^+ = \frac{(E_w^+)^{3/2}}{L\sqrt{\rho}}, \tag{39}$$

where $L = C/\sqrt{B}$ is the correlation length of the Alfvén waves and C is an adjustable input parameter. A similar dissipation is used for the '−' waves. These Alfvén wave equations assume isotropic ion pressure. We further assume that the wave dissipation will heat the ions, $Q_i = Q^+ + Q^-$, and the pressure associated with these waves

$$p_w = (\gamma_w - 1)E_w \tag{40}$$

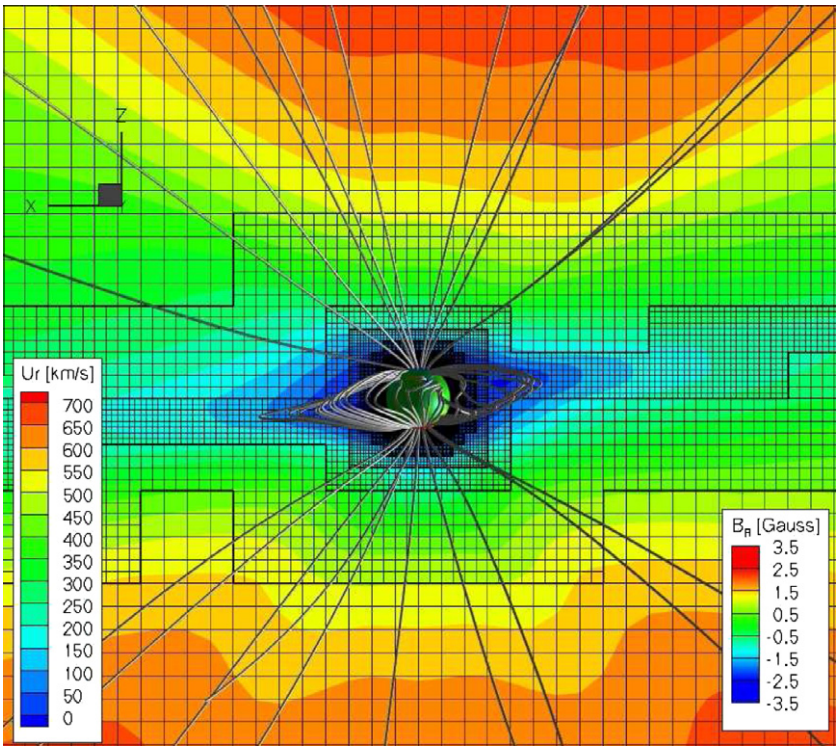

Fig. 8. 3D solar wind simulated with BATS-R-US using Alfvén waves, electron pressure equation and heat conduction. The radial velocity in the *xz*-plane shows the bimodal fast-slow wind due to the Alfvén waves. A few selected field lines depict the helmet streamer and coronal holes. The radial magnetic field is shown on the surface of the Sun.

will accelerate the solar wind plasma. Here $E_w = E_w^+ + E_w^-$ is the total wave energy density. The momentum and energy equations, (17) and (23), are modified to

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot \left[\rho \mathbf{u}\mathbf{u} + I\left(p + p_e + p_w + \frac{1}{2}B^2\right) - \mathbf{B}\mathbf{B} \right] = 0, \tag{41}$$

$$\frac{\partial e}{\partial t} + \nabla \cdot \left[(e + p + p_e + p_w + B^2/2)\mathbf{u} - \mathbf{B}\mathbf{B} \cdot \mathbf{u} + \mathbf{q}_e + (E_w^+ - E_w^-)\mathbf{u}_A \right] = 0, \tag{42}$$

where the total energy density $e = \varepsilon + \varepsilon_e + B^2/2 + E_w$ consists of the total hydrodynamic ion energy, the internal electron energy, magnetic energy, and wave energy density. For convenience, we have neglected the Hall and Biermann battery in these conservation laws. Also not shown are sources due to gravity, Coriolis, and centrifugal forces.

In Fig. 8, we show a 3D solar wind MHD simulation with the Alfvén waves, electron pressure equation, and heat conduction using the HLLE solver [73,74]. The obtained steady state was performed on a domain $|x|, |y|, |z| \leqslant 24R_{\rm Sun}$ with five levels of refinement using $4 \times 4 \times 4$ Cartesian blocks. The cell sizes vary from $1/43R_{\rm Sun}$ near the Sun to $0.75R_{\rm Sun}$ near the outer boundary. Two additional levels of refinement are used near the heliospheric current sheet (see the AMR grid in the figure). The final number of grid cells is 2.5 million.

The inner boundary is taken at 1.035 solar radii. If the center of a grid cell is inside this radius, it is regarded as a 'false' cell, and it is not updated. The boundary conditions are applied on the cell faces between 'true' and 'false' cells. Although this procedure corresponds to a ragged surface and the scheme is only first order accurate at the inner boundary, the results remain acceptable for strongly magnetized bodies, because the magnetic field suppresses flows parallel to the surface. For non-magnetized bodies the use of spherical grids is preferable, because then the resulting inner boundary surface is a perfect sphere, and one can apply second order accurate boundary conditions using regular ghost cells.

The Alfvén waves applied at the inner boundary drive the bi-modal nature of the solar wind with the fast wind in the polar region and slow wind near the equator. The few selected field lines show the location of the helmet streamer with the closed field lines and the coronal holes with some open field lines. See [35] for more detail.

2.3.7. Multi-species and multi-material equations

It is often necessary to distinguish different ion species in space plasmas, because they have different properties with respect to photoionization, collisions with neutrals, charge exchange, etc. If the plasma is collisional then the velocities

and temperatures of the ion species are approximately the same. The multi-species plasma can be modeled by solving additional continuity equations for each species indexed by $s = 1 \ldots N_{species}$:

$$\frac{\partial \rho_s}{\partial t} + \nabla \cdot (\rho_s \mathbf{u}) = S_{\rho_s}, \tag{43}$$

where ρ_s and S_{ρ_s} are the mass density and the source term for species s, respectively. Since we also solve for the total density ρ, at the end of each time step we may either replace ρ with $\sum_s \rho_s$, or we may adjust the species densities to add up to ρ.

In some applications of BATS-R-US (not in space physics) we need to distinguish multiple materials that do not mix with each other, but differ from each other in their properties, e.g. the equation of state. Here we may use the multi-species equations by simply solving for the density of each material. The numerical diffusion will eventually result in areas where there are multiple species with positive densities. As long as there is a dominant species we can use the corresponding material properties. At the material interfaces, however, the species densities can become comparable. Here we may use some weighted average of the material properties. An alternative approach is to use levelset functions which go through zero smoothly at the material interfaces. The levelset functions are initialized as the signed distance from the interface: d_s is positive inside material s and negative outside. The d_s functions satisfy simple advection equations

$$\frac{\partial d_s}{\partial t} + \nabla \cdot (d_s \mathbf{u}) = d_s \nabla \cdot \mathbf{u}. \tag{44}$$

Numerical diffusion may still result in regions where multiple d_s functions are positive or where they are all negative, but due to the smoothness of the distance functions d_s the numerical diffusion is less than for the discontinuous density functions ρ_s.

2.3.8. Multi-fluid MHD

When the different ion and neutral species are collisionless, their velocities and temperatures can differ significantly. In this case we need a multi-fluid description, where each fluid has its own density, velocity and temperature. Here we briefly describe the multi-fluid MHD equations that are implemented in BATS-R-US. For the derivation of the multi-ion MHD equations see [44].

We allow an arbitrary number of ion and neutral fluids indexed by $s = 1 \ldots N_{ion}$ and $n = N_{ion} + 1 \ldots N_{fluid}$, respectively. For the ion fluids the following equations are solved

$$\frac{\partial \rho_s}{\partial t} + \nabla \cdot (\rho_s \mathbf{u}_s) = S_{\rho_s}, \tag{45}$$

$$\frac{\partial \rho_s \mathbf{u}_s}{\partial t} + \nabla \cdot (\rho_s \mathbf{u}_s \mathbf{u}_s + I p_s) = n_s q_s (\mathbf{u}_s - \mathbf{u}_+) \times \mathbf{B} + \frac{n_s q_s}{n_e q_e} (\mathbf{J} \times \mathbf{B} - \nabla p_e) + S_{\rho_s \mathbf{u}_s}, \tag{46}$$

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \times \left(-\mathbf{u}_e \times \mathbf{B} - \frac{\nabla p_e}{q_e n_e} \right) = 0, \tag{47}$$

$$\frac{\partial p_s}{\partial t} + \nabla \cdot (p_s \mathbf{u}_s) = -(\gamma - 1) p_s \nabla \cdot \mathbf{u}_s + S_{p_s}, \tag{48}$$

$$\frac{\partial p_e}{\partial t} + \nabla \cdot (p_e \mathbf{u}_e) = -(\gamma - 1) p_e \nabla \cdot \mathbf{u}_e + S_{p_e}, \tag{49}$$

where n_s, q_s, $\mathbf{u}_s$ and p_s are the number density, electric charge, velocity and pressure for the ion fluid s, respectively. The charge averaged ion velocity

$$\mathbf{u}_+ = \frac{\sum_s q_s n_s \mathbf{u}_s}{q_e n_e}, \tag{50}$$

defines the average velocity of the positive current carriers, so the electron velocity can be written as

$$\mathbf{u}_e = \mathbf{u}_+ - \frac{\mathbf{J}}{q_e n_e}. \tag{51}$$

The electron number density can be obtained from charge neutrality $q_e n_e = \sum_s n_s q_s$. The source terms S_{ρ_s}, $S_{\rho_s \mathbf{u}_s}$, S_{p_s} and S_{p_e} are due to various processes, including gravity, charge exchange, recombination, photo-ionization, etc. In the above equations the resistivity is neglected. One may also drop the Hall and Biermann battery terms in the induction Eq. (47) and the electron velocity Eq. (51) to avoid the stiffness due to the whistler waves. In this case $\mathbf{u}_e = \mathbf{u}_+$.

The multi-ion MHD Eqs. (45)–(49) cannot be written in conservation form because of the n_s/n_e multipliers on the right hand sides of the momentum Eq. (46). To improve the conservative properties of the scheme we allow to also solve for the total momentum density $\rho u = \sum_s \rho_s u_s$ which obeys a conservation law. At the end of each time step the individual momenta $\rho_s \mathbf{u}^*$ can be adjusted so that they add up to the total momentum $\rho \mathbf{u}^{n+1}$. With respect to energy conservation, there is no simple equation for the total energy density. We have the option to solve for the hydrodynamic energy densities $e_s = \rho_s u_s^2/2 + p_s/(\gamma - 1)$ as

$$\frac{\partial e_s}{\partial t} + \nabla \cdot [(e_s + p_s)\mathbf{u}_s] = \mathbf{u}_s \cdot \left[n_s q_s (\mathbf{u}_s - \mathbf{u}_+) \times \mathbf{B} + \frac{n_s q_s}{n_e q_e} (\mathbf{J} \times \mathbf{B} - \nabla p_e) \right] + S_{e_s}, \tag{52}$$

instead of the ion pressure Eq. (48). As long as the electron pressure and the magnetic energy density are small relative to the kinetic energy density, the jump conditions will be approximately correct across a shock wave. This is approximately true for the bow shocks around planets.

Finally we also have to cope with the issue of positivity. In the multi-species case a small negative ρ_s, while physically incorrect, may not cause any numerical problems (except for some source terms), and it can be easily corrected if necessary. In the multi-fluid case we have to maintain positivity of ρ_s and p_s otherwise the wave speeds become imaginary. In regions where some fluids have very small density relative to the total ρ, we maintain a lower limit on ρ_s (some small fraction, e.g. 10^{-4}–10^{-9}, of ρ) and set the velocity $\mathbf{u}_s$ and temperature $T_s = p_s/(k_B n_s)$ of the minor fluid to the average velocity and temperature of all ion fluids combined. This is a physically meaningful state that can interact correctly with regions where the same fluid occurs with significant density. See [44] for further details on multi-ion MHD.

The neutral fluids satisfy the Euler equations with source terms:

$$\frac{\partial \rho_n}{\partial t} + \nabla \cdot (\rho_n \mathbf{u}_n) = S_{\rho_n}, \tag{53}$$

$$\frac{\partial \rho_n \mathbf{u}_n}{\partial t} + \nabla \cdot (\rho_n \mathbf{u}_n \mathbf{u}_n + I p_n) = S_{\rho_n \mathbf{u}_n}, \tag{54}$$

$$\frac{\partial p_n}{\partial t} + \nabla \cdot (p_n \mathbf{u}_n) = -(\gamma - 1) p_n \nabla \cdot \mathbf{u}_n + S_{p_n}, \tag{55}$$

and we also solve the energy equation

$$\frac{\partial e_n}{\partial t} + \nabla \cdot [(e_n + p_n)\mathbf{u}_n] = S_{e_n}, \tag{56}$$

for the total neutral fluid energy density $e_n = \rho_n u_n^2/2 + p_n/(\gamma - 1)$.

The ion-fluids are strongly coupled by the magnetic field, so we calculate the numerical flux function (e.g. Rusanov or HLLE) for all the ion fluids together. In particular, the fastest wave speeds are estimated as the maximum (minimum) of the fast magnetosonic speeds $u \pm c_{fast}$ for the total ion fluid (with total mass density, momentum and pressure) and the sound wave speeds $u_s \pm c_s$ of the individual ion fluids. In addition, we use a point-implicit scheme to evaluate the terms proportional

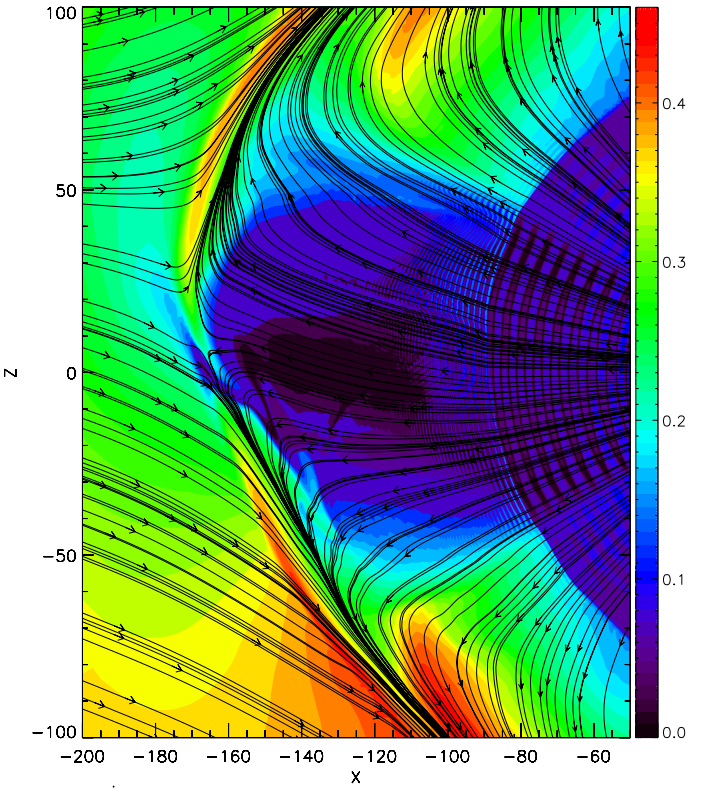

Fig. 9. Part of the $y = 0$ cut of a 3D outer heliosphere simulation with BATS-R-US using 1 ion and 4 neutral fluids. The coordinates are in astronomical units. The magnetic field strength [nT] is shown in color, the streamlines indicate the flow field. Note how the fine ripples of the current sheet are compressed as the solar wind crosses the termination shock.

to $(\mathbf{u}_s - \mathbf{u}_+)$ on the right hand sides of the momentum and energy density Eqs. (46) and (52), which may become numerically unstable with a simple explicit evaluation.

The neutral fluids, on the other hand, are typically weakly coupled to the ions and each other, so one can use separate wave speeds for each neutral fluid in the numerical scheme. BATS-R-US also allows using different numerical flux functions for the ions and the neutrals.

An interesting multi-fluid application is the outer heliosphere, where the interstellar neutrals can charge exchange with the solar wind and the shocked interstellar plasma fluid. Depending on the region where the charge exchange happens, the neutrals will have different velocities and temperatures. Since the neutrals can move freely across the magnetic field, the resulting neutral distribution cannot be described with a single Maxwellian fluid. A reasonable approximation is to use multiple neutral fluids (or populations), each with its own density, velocity and pressure [39]. Fig. 9 shows a most recent simulation of the interaction of the solar wind with the interstellar material and magnetic field. This time accurate simulation requires 1.4 billion (1.4×10^9) cells ranging from 0.03 to 31.25 AU in size (10 levels of grid refinement). The computational domain extends from $r = 30$ AU to $|x|, |y|, |z| < 1000$ AU, and the simulation took 230,000 time steps so far. BATS-R-US ran on about 2000 CPU cores of the Pleiades supercomputer for several weeks. The fine grid is required to resolve the very small (about 0.7 AU) separation of the sectors of the current sheet in the heliosheath. The sectors are formed due to the tilt of the solar magnetic field relative to the rotation axis of the Sun. The scientific significance of the results will be discussed in another publication.

2.3.9. Radiative transfer

For the sake of completeness we mention that BATS-R-US can now also solve for radiation transfer in the flux limited multi-group diffusion approximation. This capability is used in high energy density applications outside the scope of space physics [75,76]. We will describe the equations and the numerical algorithms in a future publication (van der Holst et al., 2011, submitted to ApJ. Suppl.).

3. Adaptive spatial discretization

The spatial discretization in BATS-R-US is based on total variation diminishing (TVD) [69,77] type numerical schemes with second and third order (in smooth monotonic regions) [70] accurate slope limiters. We have developed a second order accurate oscillation free interpolation scheme for the grid resolution changes [78]. A large variety of numerical flux functions are available, including the local Lax-Friedrichs or Rusanov [79], HLLE [73,74], Artificial Wind [80], HLLD [81], Roe [82] and Godunov [83] fluxes. The Rusanov, HLLE and Artificial Wind schemes work for all systems of equations, the HLLD and Roe schemes are implemented for ideal MHD only, while the Godunov scheme (exact Riemann solver) can be used for hydrodynamics. The optimal combination of flux functions, limiters, and divergence control methods is highly application dependent, and having a large number of choices can be regarded as an algorithmic adaptivity.

BATS-R-US was designed around a 3D block-adaptive grid. Each block has the same number of cells, but the blocks can have different sizes in physical space. The blocks can be split into eight children (refinement), or eight blocks can be merged into one (coarsening). While cell-based adaptivity offers the most flexible grid structure, block adaptivity has a lot of advantages: each block is a simple structured grid allowing simple numerical schemes, the fixed sized blocks are easy to load balance, the data corresponding to a block can easily fit into the cache, and the fixed length loops can be efficiently optimized (unrolled) by the compiler [84].

The block-adaptive grid algorithm has been improved and extended a lot over the years. It has been generalized to handle arbitrary number of variables and generalized coordinates. We implemented message passing over the poles of spherical and cylindrical grids, added message passing of cell faces edges and corners, etc. We wrote various algorithms, including block-to-block as well as single buffer communication. Recently, however, we faced a new challenge: generalizing the AMR algorithms to two spatial dimensions. Since the original BATS-R-US algorithms have been hard coded for three spatial dimensions, generalization to 2D (and 1D) would have required an almost complete rewrite. We have decided that we might as well start from scratch and design a new, separate Block Adaptive Tree Library, BATL.

We will first discuss block-adaptive grids with generalized coordinates, then the new AMR library, BATL.

3.1. Generalized coordinates

Generalized coordinates allow a continuous mapping from a logically Cartesian coordinate system into a general curved system. Simple and practically important examples are stretched Cartesian grids, as well as cylindrical, spherical and toroidal coordinates. In BATS-R-US we implemented the generalized coordinates in combination with Cartesian vector components. This means that the governing equations are kept in the Cartesian form, only the geometry of the grid is modified. This choice allows a general implementation as well as avoids singularities along the symmetry axis of cylindrical and spherical coordinates.

The connectivity of grid cells across the symmetry axis of cylindrical and spherical grids is complicated. To keep the algorithm simple, resolution changes are allowed along but not around the symmetry axis (see [15] for a more general algorithm that allows resolution changes around the axis). An additional problem is that the cells around the symmetry axis become

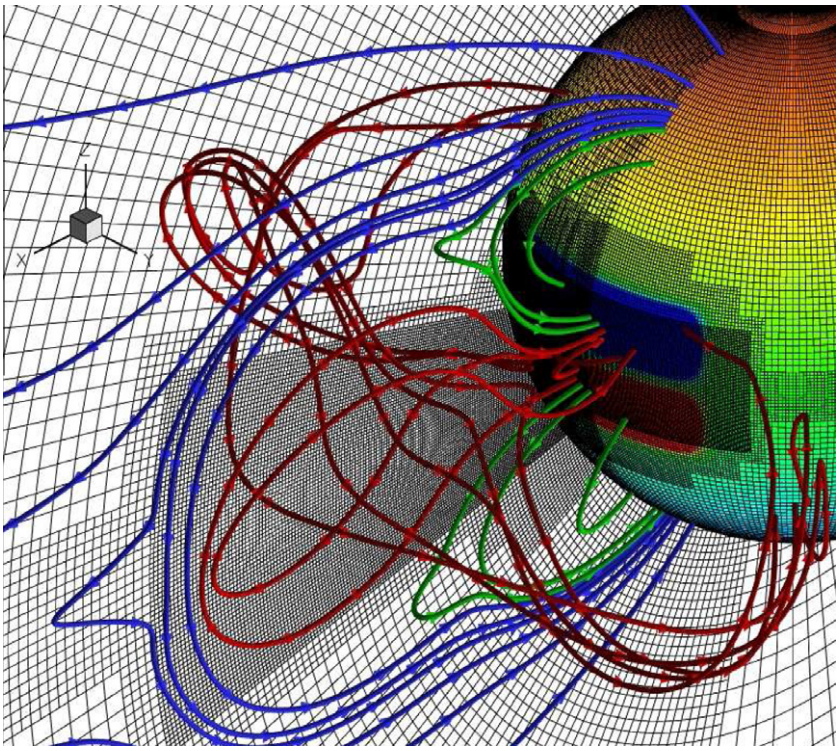

Fig. 10. A few selected field lines of the breakout CME eruption. The colors represent the various initial magnetic flux systems: central breakout arcade in red, neighboring arcades in green, and the overarching helmet streamer and open field lines in blue. The mesh in the xz and $r = R_{\rm Sun}$ plane demonstrate the spherical AMR grid. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

very thin. For an explicit time integration scheme, this can severely limit the time step. We have developed a simple yet conservative 'supercell' algorithm which averages the cells around the symmetry axis after every time step. To make the scheme second order accurate, a limited least-squares slope is calculated from the cells surrounding the supercell, and the cell values inside the supercell are filled accordingly. See Appendix B for more detail.

Spherical coordinates can be very useful for space science simulations, since the grid will then be aligned with the surface of the Sun, planets, or moons. Fig. 10 shows a simulation of a Coronal Mass Ejection (CME) from the Sun using the breakout mechanism [28,29]. An elongated arcade is added in the closed field line region of the steady state solar wind. This arcade mimics a quiescent filament and the orientation is such that there is a magnetic null on the leading edge. The simulation was performed on a spherical grid extending over the range $R_{\rm Sun} \leqslant r \leqslant 24R_{\rm Sun}$, $0 \leqslant \theta \leqslant \pi$, and $0 \leqslant \varphi \leqslant 2\pi$. The grid is constructed as follows: We start with a grid consisting of $128 \times 128 \times 256$ cells, partitioned in $4 \times 4 \times 4$ blocks. The mesh is uniform in the angle directions, but uses a logarithmic stretching in the radial coordinate. Near the poles, the resolution is decreased by one AMR level and in addition the supercell algorithm is used to avoid too small time steps. Near the heliospheric current sheet of the solar wind, the resolution is increased by one AMR level. The initial CME eruption phase is accurately captured by increasing the number of AMR levels by two in the region $(r,\theta,\varphi) \in [R_{\rm Sun}, 3.14R_{\rm Sun}] \times [-14^\circ, 14^\circ] \times [-36^\circ, 36^\circ]$. The final grid consists of 134,752 $4 \times 4 \times 4$ blocks yielding a total of 8.6 million mesh cells with the smallest cell size at the bottom of the arcade, where $(\Delta r, \Delta\theta, \Delta\varphi) \approx (0.0062R_{\rm Sun}, 0.35^\circ, 0.35^\circ)$. By applying shear flow near the polarity inversion line of this magnetic arcade, the field near the null starts to reconnect and removes the overlying helmet streamer field, resulting in a run-away breakout CME eruption.

3.2. Block Adaptive Tree Library – BATL

We decided to write BATL in Fortran 90 using the MPI library for communication, so it requires the same software environment as BATS-R-US or the SWMF. We also considered using the LASY precompiler [85] that allows writing code for arbitrary number of spatial dimensions, as it was done in MPI-AMRVAC (see Keppens et al., in this issue), but we opted to write the code in plain F90 so that the source code is easy to read for other developers. We also decided to avoid the use of pointers and derived types, although they are definitely suitable for representing the dynamic and complex tree structure of an AMR grid (these language elements were heavily used in the original BATS-R-US code as well as in MPI-AMRVAC). The reason for this choice has to do with the limitations of the MPI libraries available on today's supercomputers: information contained in pointers or derived types cannot be passed. Therefore we use simple integer indexes instead of pointers, and arrays with

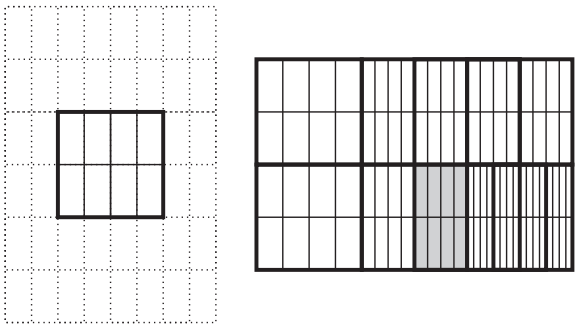

Fig. 11. A 2D block adaptive grid. The left panel shows a single block (thick square) with $n_1 = 4$ times $n_2 = 2$ cells (thin solid lines), and $n_G = 2$ layers of ghost cells (dotted lines). The right panel shows an adaptive grid with $N_1 = 3$ times $N_2 = 2$ root blocks. Only the first dimension is refined, so the refinement ratios are $r_1 = 2$ and $r_2 = 1$. The shaded block is at level $l = 1$ and its integer coordinates are $c_1 = 4$ and $c_2 = 1$.

named indexes instead of derived types. This makes the code simpler, more efficient and still flexible enough. The whole AMR grid structure can be described by a single integer array. For example the index of the second child of a tree node is stored as `iTree_IA(Child2_,iNode)` while the AMR level is stored as `iTree_IA(Level_,iNode)` where `Child2_` and `Level_` are named indexes, i.e. integer constants. The array name `iTree_IA` follows our naming standard: the initial `i` indicates that it is of type integer, the `_IA` suffix indicates that it is an array indexed by some general index (`I`) followed by a node index (`A`). This integer array is easy to communicate through MPI or to save into a file for restart.

3.2.1. Grid structure

BATL supports 1, 2 or 3 dimensional block-adaptive grids, where every grid block has $n_1 \times n_2 \times n_3$ cells. In 2D grids $n_3 = 1$, while in 1D grids $n_2 = n_3 = 1$. The n_1, n_2 and n_3 constants (integer parameters in terms of F90) are set by a `Config.pl` script before BATL is compiled. This makes the 1D and 2D applications efficient, because the operations related to the ignored dimensions are eliminated at compile time. Due to the restrictions of Fortran 90, the arrays containing various data for the blocks always have three spatial indexes, but the last one or two dimensions may have a single element. The grid blocks are surrounded by n_G layers of ghost cells to facilitate information exchange between the blocks. The value of the n_G constant is set before compilation for the sake of efficient optimization. For 2D and 1D grids no ghost cells are required in the ignored dimensions. In the used dimensions the cell indexes go from $1 - n_G$ to $n_i + n_G$. The left panel of Fig. 11 shows a block with $n_1 = 4$, $n_2 = 2$, $n_3 = 1$ and $n_G = 2$.

In BATL the grid refinement ratio can be 2 only, on the other hand not all dimensions have to be refined. For each dimension where $n_i > 1$ the refinement ratio r_i can be 1 or 2. If $r_i = 1$ then dimension i cannot be refined or coarsened, if $r_i = 2$, there is refinement in dimension i. This means that every block can have $r_1 r_2 r_3 = 2$, 4 or 8 children blocks. The r_1, r_2 and r_3 constants are also set by the `Config.pl` script before compilation. A possible application of partial AMR can be a spherical grid where the refinement is done in latitude and longitude only, but not in the altitude (radial) direction. In the refined dimensions the number of cells n_i has to be an even number and $n_i \geqslant 2n_G$ must hold so that the ghost cells of a coarser neighbor are covered by the finer block. These restrictions arise from the ghost cell filling procedures at resolution changes that we will discuss below. The right panel of Fig. 11 shows an adaptive grid that is refined in the first dimension only: $r_1 = 2$ and $r_2 = 1$.

At the lowest refinement level the root blocks are arranged into an $N_1 \times N_2 \times N_3$ root grid. The N_1, N_2 and N_3 integers are set during run time so the shape of the root grid can be changed without recompiling BATL. Since the root grid size occurs in the outer loops only, making them a constant would not improve the efficiency of the code significantly. In 2D grids $N_3 = 1$, while in 1D grids $N_2 = N_3 = 1$. Starting with each root block the refinement creates a binary tree, quadtree or octree down to the finest level of blocks.

The topology of the adaptive grid is described by a set of *treenodes*, or nodes, for short. The nodes are identified by a global node index. All nodes have the following information stored in the `iTree_IA` integer array:

- the status of the node (used, unused, to be refined, to be coarsened, etc.);
- the current, the maximum allowed and minimum allowed AMR levels for this node;
- the three integer coordinates with respect to the whole grid;
- the index of the parent node (if any);
- the indexes of the children nodes (if any);
- the processor index where the block is stored for active nodes;
- the local block index for active nodes.

The three integer coordinates c_i specify the location of the node with respect to a virtual uniform grid of the given level. At level l the ranges of integer coordinates are $c_i = 1 \ldots N_i r_i^l$, where $l = 0$ corresponds to the root level. The maximum number of levels is limited to 30 because we use 32-bit signed integers to store the coordinates. This is more than sufficient for our

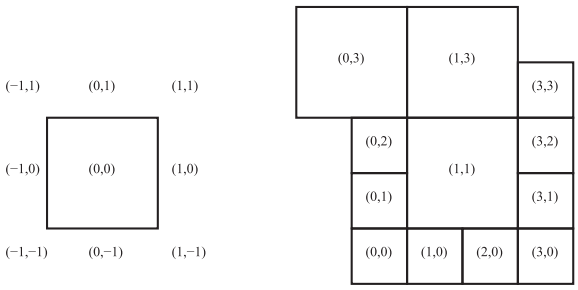

Fig. 12. The left and right panels show the indexing of the neighbor directions, and neighbor blocks, respectively, in 2D.

current applications. The c_i coordinates are useful for many purposes: the parity of c_i specifies the location of the node with respect to the parent block, one can easily find blocks that are at the edges of the computational domain, and the (generalized) physical coordinates can be easily calculated from the integer indexes.

The node information (18 integers per node in 3D) is replicated on all processors, which simplifies the algorithms substantially, and it allows a lot of functionality without inter-processor communication. As long as the total number of nodes does not exceed a few million, storage should not be an issue on most supercomputers with a gigabyte or more memory per core. If it becomes necessary, the node information could be distributed among the processors to reduce the storage requirements.

To speed up the information exchange between the grid blocks, we precalculate neighbor information. For each block (active node) we store the AMR level difference with respect to the neighbors in all directions, including diagonal directions. In 3D there are 26 directions, but it is easiest to store 27 integers in a $3 \times 3 \times 3$ array. We also store the node indexes of all the neighbors in a $4 \times 4 \times 4$ array, where indexes 0 and 3 correspond to the neighbors on the two sides, while indexes 1 and 2 correspond to the two halves of the block which is needed to distinguish the finer neighbors in the tangential direction. Fig. 12 shows how the directions and neighbors are assigned to indexes in 2D. For the sake of storage reduction, the neighbor information is distributed over the processing elements, and it is available for the local blocks (active nodes) only. Storing $27 + 64 = 91$ integers per block is a minor overhead compared to the other data associated with grid blocks.

The neighbor information can be obtained by traversing the tree up and down. This results in an efficient but complicated algorithm. An alternative approach that we use in BATL relies on the integer block indexes and the efficient binary search algorithm of finding the block that covers a point. For each block we loop through all the possible neighbors (see Fig. 12) and generate a point position (in normalized generalized coordinates) that should lie inside that neighbor. We take into account periodicity and/or poles of a spherical/cylindrical grid by appropriately shifting the point position. Then we use the binary search algorithm to find the block that contains the point. This scheme is relatively simple and still quite efficient.

3.2.2. Grid geometry

Up to this point, we kept the discussion of BATL fully general with respect to the grid geometry. Indeed, BATL is intended to support arbitrary grid geometries, including Cartesian, spherical, cylindrical, toroidal, or even arbitrarily stretched grids. The whole computational domain is a brick in the *generalized coordinates*, but this can correspond to a spherical shell, a cylinder, or a distorted torus in real space. Similarly, every grid block is uniform in the generalized coordinates, but not necessarily in the Cartesian space. For the sake of efficiency we precompute useful geometrical information: cell sizes (in generalized coordinates), face areas, face normals, cell volumes as well as the cartesian coordinates of the cell centers. This information is distributed over the processors, and it is indexed by the local block index.

In each dimension, one can either use boundary conditions (usually implemented by setting the ghost cells) or the dimension can be set to be periodic. Periodicity is simply implemented by setting the neighbors of the block to the opposite side of the grid. This works for the azimuthal coordinate of cylindrical and spherical grids, as well as for toroidal and poloidal coordinates of toroidal grids. Grid geometries that have a singular pole (e.g. spherical and cylindrical grids) require special message passing across the polar axis. Alternatively, the cells surrounding the poles can be merged into a supercell. All of these algorithms are fully implemented into the 3D AMR code of BATS-R-US, but the current implementation of the new BATL code is restricted to 1, 2 and 3D Cartesian and axially symmetric 2D $R - Z$ geometries.

3.2.3. Refinement, coarsening and load balancing

The grid is adapted in the following stages:

1. Based on some criteria assign the blocks for refinement or coarsening.
2. Enforce proper nesting criteria.
3. Coarsen tree by deactivating children nodes and activating the parent node.
4. Refine tree by adding new active children nodes and deactivating the parent node.
5. Load balance the nodes by assigning them to the processors.

6. Restrict, load balance and prolong grid block data.
7. Remove the coarsened tree nodes and compact the tree.

A block is refined if any of its siblings require refinement, or the nesting criteria require its refinement and it has not reached the maximum allowed level. The block is coarsened if all the siblings allow the coarsening, the block has not reached the allowed minimum level and the coarsening does not contradict the proper nesting. The proper nesting requires that the AMR levels of neighboring grid blocks can only differ by one. We apply this restriction to neighbors in all diagonal directions so that we can keep the message passing algorithm relatively simple.

For purposes of load balancing the active nodes, i.e. blocks, are ordered by a space filling curve. The root nodes are simply ordered in the usual array order by looping over them dimension by dimension. The trees starting from the root nodes are ordered recursively by looping over the children again dimension by dimension. This ordering corresponds to the Morton space filling curve. Another popular choice that we actually use in the original BATS-R-US code is the Peano-Hilbert space filling curve. It was found that Morton and Peano-Hilbert space filling curves provide similar data locality, and the Morton ordering is much simpler. In the simplest case, load balancing means that we cut the space filling curve into N_p equal pieces, where N_p is the number of processors. BATL also supports a more complicated load balancing procedure, when there are different types of blocks, and each type is distributed evenly among the processors. For example the blocks on different AMR levels, or explicitly and implicitly advanced blocks can be load balanced separately. To do this the space filling curve is *colored* according to the block type, and then the curve of a given color is cut into N_p equal pieces. In general the number of blocks per type is not an integer multiple of N_p, but we ensure that the number of blocks of a given type as well as the total number of blocks per processor varies by at most 1.

The data is restricted (coarsened) by averaging the 2, 4 or 8 fine cells into a coarse cell. On non-Cartesian grids the volumes of the fine cells are taken into account, so that the coarsening is conservative. This is a second order accurate procedure in the finite volume sense. We copy/send the already restricted data to minimize communication.

The data is prolonged (refined) by interpolating the coarse data onto the fine grid. To make the procedure conservative and oscillation free, we calculate the gradients in each coarse cell with central differencing, and then limit the slopes with the monotonized central limiter with an adjustable β parameter. This requires that the coarse data is sent/copied with one extra layer of ghost cells to the processor where the prolongation is done. On non-Cartesian grids, the coarse data is multiplied by the coarse cell volume V_c before calculating the slopes, and the interpolated fine data is divided back by the fine cell volume times the total refinement ratio $V_f r_1 r_2 r_3$ at the end.

The coarsening, load balancing and prolongation are done by looping over the nodes three times. The actual work is done only by the processors that send and/or receive the data, so the expensive part of the algorithm is done in parallel. The MPI communication is done block by block, so we can fill up the holes created by sending data to another processor with the received data. This minimizes, but does not eliminate, the number of holes in the arrays containing the block data. We use a logical array `Unused_B` to keep track of the local block indexes that are not used. This array can also be used to switch on and off blocks for various time stepping schemes.

After the load balancing of the data is done, the tree nodes that contained the coarsened blocks can be removed. This will create holes in the tree data array. Since the tree data is relatively small, we can remove the holes by *compacting* the tree: the used nodes are moved next to each other, while the indexes of the parent and children nodes are corrected.

3.2.4. Message passing

Message passing fills in the ghost cells of the blocks based on information taken from the neighboring blocks. When the neighbor block is at the same grid level, the ghost cells are simple copies of the normal cells of the neighboring block. When the neighbor is at a finer or coarser level, the ghost cell is filled in with restriction and prolongation operations, respectively, similar to the coarsening and refinement steps. Here, however, conservation of variables in the ghost cells is not necessary. Conservation of the overall scheme is ensured by the face flux correction step [86]. On the other hand it is important to ensure that no spurious oscillations form at the resolution changes. This requires careful prolongation schemes [78].

Message passing of ghost cells is done at least once every time step (or iteration), and it involves a lot of data movement, so it is important to make it efficient. Ghost cells are filled in for various purposes. In some cases it is sufficient to fill in the ghost cells in the up to six principal directions only, in other cases the diagonal (edge and corner) ghost cells have to be filled in as well. The number of ghost cell layers that need to be filled in can vary from 1 to n_G. For some time discretization schemes only a subset of the blocks need to be updated with new ghost cell information. When the grid is coarsened or refined, the list of blocks and the location of resolution changes gets modified. For some purposes it is necessary to use second order accurate restriction and prolongation, for others first order, or special prolongation and restriction procedures are needed. BATL has to provide all this functionality in 1, 2, or 3 dimensions.

For the sake of efficiency all the data sent from processor A to processor B is collected into a single buffer, and then all the buffers are sent with non-blocking MPI send/receive communications. For second order accurate prolongation it is necessary to fill in the ghost cells with same or finer underlying cells first, so that there are ghost cells available for the prolongation operation. Then the prolonged ghost cells are message passed in a second stage.

The remaining task is to pack and unpack the buffer. In the original BATS-R-US algorithm the order of blocks and the ranges of sent cells and received ghost cells are precalculated and saved into integer arrays. This allows sending the data

only, which minimizes the size of the MPI messages. On the other hand, every time the communication pattern changes (e.g. due to AMR), the index arrays have to be recalculated.

In BATL a different algorithm is used: we store the index of the receiving block and the cell index ranges into the buffer together with the data. While this slightly increases the buffer size, it greatly simplifies the algorithm. The data to be sent is calculated the same place as the receiving block index and cell index ranges, and all this information is stored into the real buffer array. The unpacking is trivial: the receiving processor reads the block index and array ranges first, and then reads the buffer data directly into place. In 2D there are five extra real numbers to send, in 3D there are seven. As long as the number of variables sent is large, this results in an acceptable overhead. The algorithm can be improved by storing the block index and the index range information into a single real number, since there are only a finite number (less than 100) possibilities for the index ranges. Then the overhead relative to the data becomes truly negligible.

We use this approach in the message passing of face centered values that is needed for the flux correction step at the resolution changes. Here we send the sum of the up to 4 fine face fluxes to the coarse neighbor, where the sum of these fluxes will replace the flux calculated on the coarse cell face. The coarse block index i_B, the block face index $i_F \leqslant 6$ and the index of the block subface $i_S \leqslant 4$ (there are up to four fine neighbor blocks sharing each face of the coarse block) are combined into a single real number $r = 100i_B + 10i_F + i_S$. This number r is inserted before the data, the face fluxes, for each face. There are at least 2 (in 2D) or 4 (in 3D) coarse cell faces and typically 8 or so variables sent per fine block face, so the overhead of sending r is below 10%.

For the sake of convenience BATL implements methods to store and correct the face fluxes in addition to the message passing.

3.2.5. Performance and scaling

We have performed parallel scaling studies on the Pleiades computer at NASA Ames using the Harpertown (Xeon E5472) nodes of the SGI ICE system connected with an Infiniband network. Fig. 13 shows the weak scaling from 1 to 16,384 CPU cores. The 3D multi-material hydrodynamics equations (see Section 2.3.7) are solved with the explicit HLL scheme and Koren's limiter. The equations of state of the three materials are obtained from lookup tables, so it takes negligible time. In essence the computational work per cell update is fairly small which makes scaling more challenging. The problem size grows proportional to the number of processors: there are 40,960 cells per processor. The number of processor cores vary from 1 to 16,384, so the largest grid contains about 671 million cells. The scaling curve is based on the timings of the first 100 time steps which varies from 23 seconds on 1 core to 71.5 and 45.5 seconds wall-clock time on 16,384 cores with the $8 \times 8 \times 8$ and $16 \times 16 \times 16$ block sizes, respectively. The $4 \times 4 \times 4$ blocks scale reasonably well up to 8000 cores (not shown). The timings do not include any I/O operations.

To test strong scaling we redo the same simulation but with a fixed problem size on 64–8,192 processor cores. The uniform grid contains 40,960 blocks with $8 \times 8 \times 8$ cells each, altogether about 21 million cells. The timings are based on the first 100 time steps. Fig. 14 compares strong scalings with three different ghost cell filling schemes: BATL with processor-to-processor communication described in Section 3.2.4, current BATS-R-US algorithm with precomputed processor-to-processor communication, and BATS-R-US with block-to-block communication. Interestingly the three schemes give very comparable results. It appears that the deviation from ideal scaling does not depend much on the communication algorithm, and it is probably determined by the hardware network characteristics.

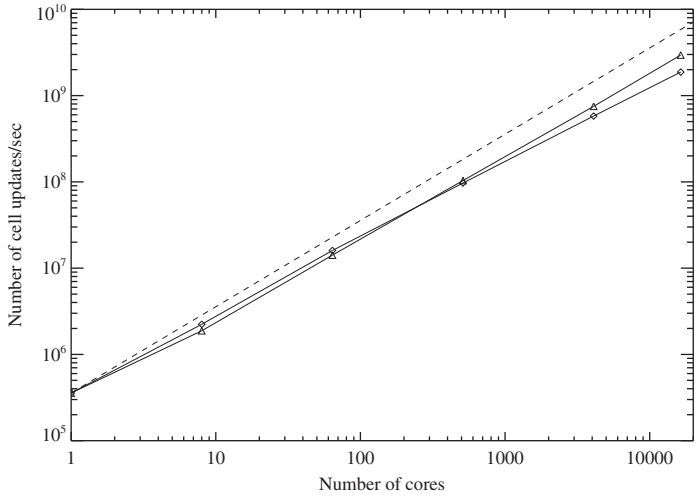

Fig. 13. Weak scaling of BATS-R-US using the BATL message passing scheme. A multi-material hydrodynamics problem is solved on a uniform grid with 40,960 grid cells per CPU core using $8 \times 8 \times 8$ (diamonds) and $16 \times 16 \times 16$ (triangles) blocks, respectively. The dashed line indicates ideal scaling.

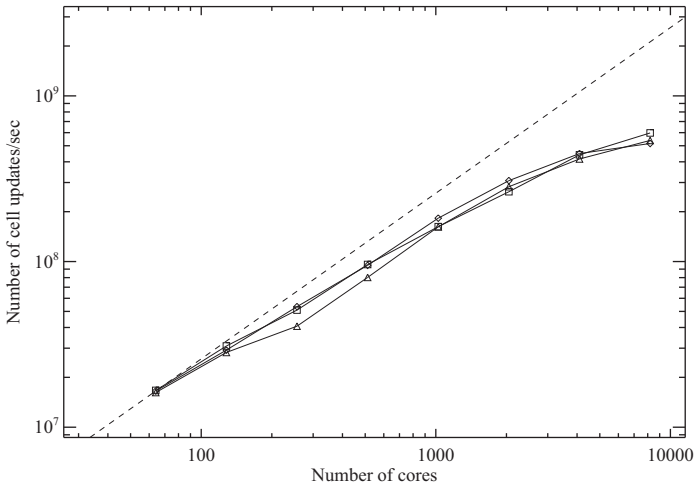

Fig. 14. Strong scaling of BATS-R-US using various message passing schemes: BATL (diamonds) versus BATSRUS message passing with processor-to-processor (triangles) and block-to-block (squares) communication. A multi-material hydrodynamics problem is solved on a uniform grid with 40,960 blocks of $8 \times 8 \times 8$ cells. The number of CPU cores range from 64 to 8,192. The dashed line indicates ideal scaling.

4. Adaptive temporal discretization

We use BATS-R-US to solve steady state as well as time accurate problems. Often the steady state solution is used as an initial condition for a time accurate simulation. Accelerating convergence towards steady state can be achieved with local time stepping as well as gradual grid adaptation. When we start from a steady state simulation, in some cases the initial evolution happens in a small fraction of the computational domain. We can substantially speed up the simulation by using a time stepping algorithm that distinguishes between steady and evolving grid cells.

In time accurate simulations the simplest time stepping scheme uses the same time step in all the grid cells with an explicit time discretization. Both the adaptive grid and the physics of the simulated problem can make this simple explicit time stepping inefficient. The adaptive grid creates cells of very different sizes, thus the CFL condition may give widely varying time step limits. The maximum propagation speed may also vary substantially in the domain. Both of these issues may be addressed by subcycling, i.e. the different cells take different number of time steps to get to the same time level.

In some cases the equations are stiff, i.e. the numerical stability restricts the time step to be much smaller than what would be dictated by accuracy considerations. Depending on the mathematical properties of the stiff terms, we can use point-implicit, semi-implicit or fully implicit time discretization schemes to allow larger time steps. We have also developed an explicit/implicit scheme that advances some of the grid blocks explicitly, and the rest implicitly.

The rest of this section briefly describes the various time stepping algorithms implemented in BATS-R-US.

4.1. Local time stepping towards steady-state solution

Convergence towards steady state can be greatly accelerated by employing different local time steps in every grid cell. The local time step is limited by the local stability conditions only. By taking the maximum possible time step (in an explicit time stepping scheme) the residual can propagate through the computational domain in fewer iterations. Formally the scheme can be written as

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \Delta t_i \mathbf{R}_i(\mathbf{U}^n), \tag{57}$$

where $\mathbf{U}_i$ is the vector of state variables, $\mathbf{R}_i$ is the discretized right hand side of the partial differential equation $\partial \mathbf{U}/\partial t = \mathbf{R}(\mathbf{U})$, and Δt_i is the local time step for grid cell i. The superscripts n and $n+1$ indicate the current and next time levels, respectively. When we reach steady state, $\mathbf{U}_i^{n+1} = \mathbf{U}_i^n$, so that $\mathbf{R}_i$ becomes zero irrespective of the value of Δt_i. This means that the discrete steady-state solution is consistent with the PDE $\mathbf{R}(\mathbf{U}) = 0$.

For the MHD equations the situation is a bit more complicated. The variation of the time step from cell-to-cell corresponds to a space dependent factor α in front of the time derivative. The discrete induction equation is therefore consistent with the following PDE:

$$\alpha \frac{\partial \mathbf{B}}{\partial t} = -\nabla \times \mathbf{E}. \tag{58}$$

Let us take the divergence of this equation:

$$\nabla \alpha \cdot \frac{\partial \mathbf{B}}{\partial t} + \alpha \frac{\partial (\nabla \cdot \mathbf{B})}{\partial t} = 0 \tag{59}$$

It is clear that $\nabla \cdot \mathbf{B}$ is not conserved if $\nabla\alpha$ is not zero! Since the $\nabla \cdot \mathbf{B} = 0$ condition is a consequence of the initial condition only, the local time stepping will lead to a steady state solution that has non-zero $\nabla \cdot \mathbf{B}$, unless we do something. Indeed, we have to use some modification of the induction equation so that the divergence free condition depends on the boundary conditions and not on the initial conditions. Possible modifications include the 8-wave scheme, the parabolic/hyperbolic cleaning, and the projection scheme. The constrained transport scheme, on the other hand, cannot be combined with the local time stepping towards steady state, because it relies on the initial condition to maintain the divergence-free magnetic field.

The current BATS-R-US code does not stop automatically when a steady state is reached to a given accuracy (although it would be easy to add this option). Such an automatic stopping condition is not guaranteed to work, because the numerical solution may never settle to a perfect steady state, but rather exhibit some very small (and insignificant) oscillations. In our practice, we set the number of iterations in advance and either monitor some global quantities (like total mass, momentum and energy) and/or compare consecutive snapshots to check for changes in the solution. If the steady state is not accurate enough, the run is restarted and more iterations are performed.

In combination with the local time stepping, we also use multiple sessions and grid adaptation to speed up the convergence towards steady state. Fig. 15 shows a few snapshots from a simulation that finds an approximate steady state solution for the magnetosphere. We solve the anisotropic MHD equations (neglecting the Hall term and electron pressure) with local time stepping. The initial condition has the solar wind values everywhere on a coarse grid with only 2 levels of grid refinement. In the first session the first order Rusanov scheme is used and the grid is refined at iterations 100, 200 and 300, and eventually the grid contains 3.3 million cells with sizes ranging from $1/8R_E$ near the inner boundary to $4R_E$ further away. After 1000 iterations the second session continues with the second order Rusanov scheme using the robust but somewhat

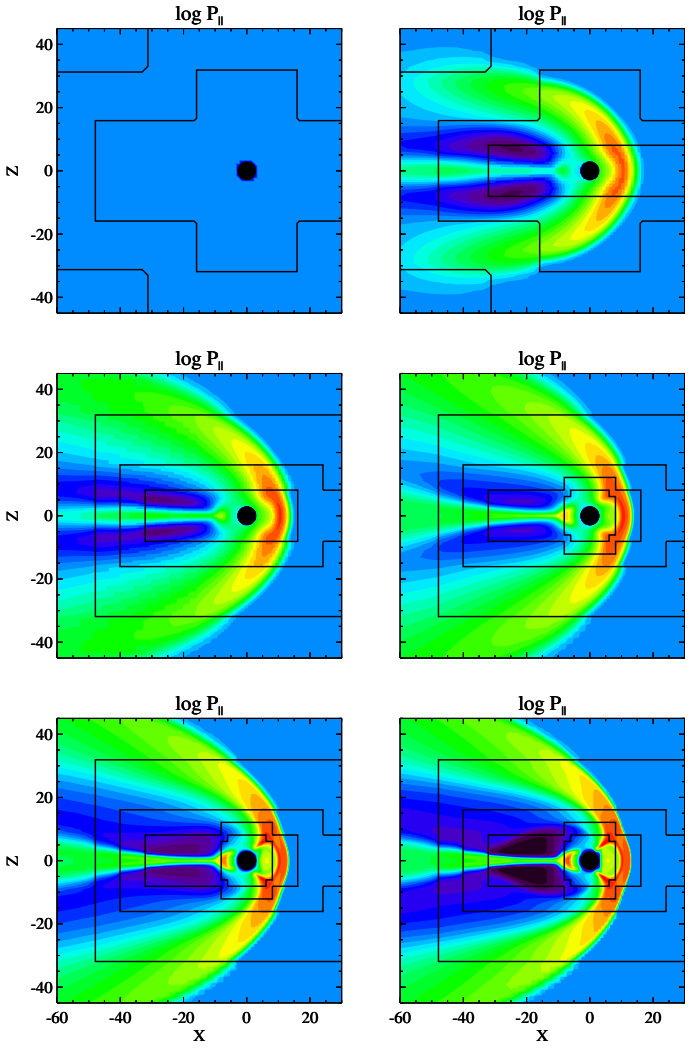

Fig. 15. Obtaining an approximate steady state solution for the magnetosphere. The six snapshots are taken at iterations 0, 100, 200, 1000, 2000, and 3000 from top left to bottom right. The parallel pressure is shown on a logarithmic scale with colors. The black lines indicate the grid resolution changes. The inner boundary indicated by the black circle is at $2.5R_E$. This close-up shows the solution near the Earth in the $y = 0$ plane.

diffusive minmod limiter. The third and final session starts at 2000 iterations, where we switch to the Artificial Wind scheme with Koren's limiter. By iteration 3000 an approximate steady state is reached that can be used as a starting point for a time-accurate run. The whole simulation takes about 6 min wall clock time using 120 cores on our Linux cluster. This run is a good example how numerical, temporal and spatial adaptivity can be combined to optimize the performance of the code.

4.2. Local time stepping for time accurate simulations

The current BATS-R-US code uses the same time step in every grid cell in time accurate mode. Another option that is often used in AMR codes is to make the time step proportional to the cell size [86]. This requires subcycling the finer levels, load balancing each grid level separately, and it also requires interpolation in time at resolution changes. The efficiency of the simpler constant time step algorithm compares well to the variable time step algorithm as long as the number of cells at the finest level is a large fraction of the total number of cells.

In many applications the fastest wave speed varies by orders of magnitude within the computational domain. For example in the magnetosphere the Alfvén speed is around 30,000 km/s near the poles of the Earth, while the fastest speed is around 500 km/s in the solar wind. Applications like this can benefit from a time stepping algorithm, where the local time step is limited by the local stability condition only.

We have implemented an algorithm in BATL that uses different time steps in different blocks, yet the whole simulation advances in a time-accurate manner. The algorithm mainly follows [87]. The basic idea is to calculate the smallest of the stable time steps over all blocks, then update a "master clock" with this time step, but advance blocks with a locally stable time step when their individual time falls behind the master clock. The ghost cells of the blocks are interpolated in time during the message passing.

For the sake of parallel efficiency the block time steps are rounded down to integer powers of two times the smallest time step. This creates relatively few groups of blocks with identical time steps (i.e. they are advanced at the same time) which makes load balancing much easier. It also makes the flux-correction step much simpler to implement than in case of arbitrary time steps that can overlap arbitrarily. In fact, the Berger and Colella scheme (that employs time steps proportional to the cell size of the AMR grid) can be regarded as a special case of the above algorithm. For constant wave speeds the two algorithms coincide.

We also plan to allow local time step adjustments during the global time step in case the stability conditions require this. This may be required if the solution changes a lot in a single global time step. This may slightly offset the load balancing, but will avoid stability issues. We have already demonstrated second order accuracy and stability of the local time stepping scheme for the pure advection equation on an AMR grid. The local CFL number is the same (0.8) as for the fixed time step algorithm. Full integration of the scheme into BATS-R-US will be done in the near future.

4.3. Partially steady-state scheme

In certain applications, a large fraction of the computational domain is in steady-state, and the time evolution is limited to a small region. An example is a CME erupting in a steady state (in the rotating frame) solar corona. To simulate these types of problems efficiently, we have developed a *partially steady-state* algorithm [20] that advances only a part of the computational grid, while the rest being in an approximate steady state is not evolved. Since BATS-R-US uses a block based grid, the computational grid is split into *changing, boundary* and *steady* blocks. The boundary blocks are at the edge of the changing domain. The changing and boundary blocks are evolved, while the steady blocks are not. As soon as a boundary block starts to change significantly (e.g. the velocity changes by more than one part in a million) it is assigned to a changing status, and its neighbors become boundary blocks. The changing and boundary blocks are load-balanced between the processors. This algorithm can speed up the simulation of the initial phase of a CME eruption by a factor of 4–6 [20].

4.4. Point-implicit scheme

If the equations contain some stiff source terms that depend on the local information only (no spatial derivatives), one can use a point-implicit scheme. Examples include chemical reactions, recombination, photo-ionization, collisional terms, and the terms proportional to $(\mathbf{u}_s - \mathbf{u}_+)$ in the multi-ion MHD Eqs. (46) and (52).

Here we describe an operator split approach. First we do an explicit update without the stiff source terms:

$$\mathbf{U}^{n+1/2} = \mathbf{U}^n + \frac{\Delta t}{2}\mathbf{R}_{\rm expl}(U^n), \tag{60}$$

$$\mathbf{U}^* = \mathbf{U}^n + \Delta t \mathbf{R}_{\rm expl}(U^{n+1/2}), \tag{61}$$

where $\mathbf{R}_{\rm expl}$ is the non-stiff part of the right hand side.

Next we add the stiff source term $\mathbf{S}^{n+1}_{\rm impl}$ to the $\mathbf{U}_{\rm impl}$ set of variables that are affected by $\mathbf{S}_{\rm impl}$ (and $\mathbf{U}_{\rm expl}$ denotes the rest of the variables):

$$\mathbf{U}^{n+1}_{\rm impl} = \mathbf{U}^*_{\rm impl} + (1-\beta)\Delta t \mathbf{S}_{\rm impl}(U^*) + \beta \Delta t \mathbf{S}_{\rm impl}(\mathbf{U}^*_{\rm expl}, \mathbf{U}^{n+1}_{\rm impl}), \tag{62}$$

which is first order in time for $\beta = 1$ and second order in time for $\beta = 1/2$. The second source term can be linearized around time level $*$:

$$\mathbf{U}_{\rm impl}^{n+1} = \mathbf{U}_{\rm impl}^{*} + \Delta t \mathbf{S}_{\rm impl}(U^{*}) + \beta \Delta t \frac{\partial \mathbf{S}_{\rm impl}}{\partial \mathbf{U}_{\rm impl}} \cdot \left(\mathbf{U}_{\rm impl}^{n+1} - \mathbf{U}_{\rm impl}^{*}\right). \tag{63}$$

The above linear equation can be solved cell-by-cell (hence the name point-implicit) for $\mathbf{U}_{\rm impl}^{n+1}$ by inverting an $N_{\rm impl} \times N_{\rm impl}$ matrix, where $N_{\rm impl}$ is the number of implicit variables. For the rest of the variables $\mathbf{U}_{\rm expl}^{n+1} = \mathbf{U}_{expl}^{*}$.

Although both the explicit E and point-implicit P operators (with $\beta = 1/2$) are second order accurate in time, the combined $E(\Delta t)P(\Delta t)$ scheme is only first order accurate unless we employ some symmetrization of the operators. One method that is particularly appealing in this particular case is Strang-type splitting, because we can take advantage of the fact that the time step in the point-implicit operator P is not limited by stability constraints. Then one can use the sequence $E(\Delta t)P(2\Delta t)E(\Delta t)$ to advance the solution by $2\Delta t$ with second order accuracy. This scheme saves one point-implicit solve relative to the first order scheme, and it is twice more efficient than the usual Strang type splitting $E(\Delta t/2)P(\Delta t)E(\Delta t/2)$. There are many other ways to achieve second order accuracy, including Godunov splitting, or using an unsplit scheme and adding the implicit stiff source term in both (60) and (61). Some other possibilities are described in [88].

The Jacobian matrix $\partial S_{\rm impl}/\partial U_{\rm impl}$ can be calculated numerically as

$$\frac{\partial S_{{\rm impl},v}}{\partial U_{{\rm impl},w}} = \frac{S_{{\rm impl},v}(\mathbf{U}^{*} + \boldsymbol{\delta}_w \epsilon_w) - S_{{\rm impl},v}(\mathbf{U}^{*})}{\epsilon_w}, \tag{64}$$

where v and w are indexes $1 \ldots N_{\rm impl}$ of the implicit variables, $\boldsymbol{\delta}_w$ is an array with all zeros except for a single one corresponding to wth implicit variable, finally ϵ_w is a small perturbation for variable w:

$$\epsilon_w = \epsilon |U_{{\rm impl},w}| + \chi_w, \tag{65}$$

where ϵ is the square root of the machine precision of real numbers (typically 10^{-6}–10^{-8} for double precision reals) and χ_w is a very small positive number relative to the typical values of $|U_{{\rm impl},w}|$, which is needed to avoid division by zero if $U_{{\rm impl},w}$ happens to be zero in a grid cell. Note that we use a positive perturbation $\epsilon_w > 0$ so that positive variables (like density or pressure) remain positive.

Numerical derivatives provide a general algorithm that works for an arbitrary stiff source term and for an arbitrary set of $\mathbf{U}_{\rm impl}$ variables. On the other hand it involves calculating the source term $N_{\rm impl} + 1$ times, which may be costly if $N_{\rm impl}$ is large and/or the source term is complicated. Since stiff source terms are implemented in the user modules, we allow the user to provide an analytic Jacobian calculation for the sake of efficiency.

4.5. Semi-implicit scheme

If the stiff terms involve spatial derivatives, the point-implicit scheme can not be applied. In this case a semi-implicit approach can be used. Examples include heat conduction (isotropic or field-aligned), resistivity, or radiative transfer in the gray or multigroup diffusion approximation. Again, only a subset of the variables $\mathbf{U}_{\rm impl}$ are affected by the stiff part of the right hand side $\mathbf{R}_{\rm impl}$.

Our semi-implicit applications typically involve Laplace operators with variable coefficients and some point-wise source terms (e.g. energy exchange between ions and electrons, or the radiation energy density). In general these terms have the following form

$$\mathbf{R}_{\rm impl} = C\nabla \cdot (\kappa \cdot \nabla \mathbf{U}_{\rm impl}) + K \cdot \mathbf{U}_{\rm impl}, \tag{66}$$

where κ and K are the diffusion and energy exchange coefficient matrices, respectively, and C is some coefficient array related to "heat capacity". The Laplace operator can be discretized with simple central differences in the uniform part of the grid, but resolution changes require special discretization to achieve second order accuracy (see [43]).

For the semi-implicit temporal discretization first we apply the explicit operator to advance the solution to time level $*$. This is an intermediate time level for $\mathbf{U}_{\rm impl}$, but it is the final update for the rest of the variables: $\mathbf{U}_{\rm expl}^{n+1} = \mathbf{U}_{\rm expl}^{*}$. Before proceeding, we linearize $\mathbf{R}_{\rm impl}^{n+1}$ by "freezing" the coefficients at time level $*$:

$$\mathbf{R}_{\rm impl}^{n+1} = C^{*}\nabla \cdot (\kappa^{*} \cdot \nabla \mathbf{U}_{\rm impl}^{n+1}) + K^{*} \cdot \mathbf{U}_{\rm impl}^{n+1}. \tag{67}$$

Now the implicit variables can be updated as

$$\mathbf{U}_{\rm impl}^{n+1} = \mathbf{U}^{*} + (1-\beta)\Delta t \mathbf{R}_{\rm impl}^{*} + \beta \Delta t \mathbf{R}_{\rm impl}^{n+1}. \tag{68}$$

This equation is a large coupled linear system due to the frozen-in coefficients and the spatial derivatives in (67). We use Krylov sub-space type iterative solvers, like Preconditioned Conjugate Gradient (PCG) (see [89] and refernces therein), BiCGSTAB [90], and GMRES [91] to solve the linear system. To make the solvers more efficient, we typically need to use a preconditioner, and solve

$$P \cdot \left(I - \beta \Delta t \frac{\partial \mathbf{R}_{\rm impl}}{\partial \mathbf{U}_{\rm impl}} \right) \cdot \Delta \mathbf{U}_{\rm impl} = P \cdot \Delta t \mathbf{R}_{\rm impl}(U^*), \tag{69}$$

where $\Delta \mathbf{U}_{\rm impl} = \mathbf{U}^{n+1}_{\rm impl} - \mathbf{U}^{*}_{\rm impl}$ and P is the (left) preconditioner operator, which is some approximation of $(I - \beta \Delta t \partial \mathbf{R}_{\rm impl}/\partial \mathbf{U}_{\rm impl})^{-1}$. Since $\mathbf{R}^{n+1}_{\rm impl}$ is linearized, calculating the Jacobian matrix elements as well as P is fairly straightforward. Typically P is applied grid block by grid block independently, which corresponds to a specific form of additive Schwarz preconditioning. Currently we use a Block Incomplete Lower–Upper decomposition (BILU) preconditioner, but we plan to explore multilevel preconditioning in the future.

We note that in the PCG algorithm the multiplication with P is part of the scheme, and cannot be written in the form of (69). The PCG algorithm works for symmetric positive definite matrices only, but we found it to be the most efficient Krylov method when applicable. GMRES works for non-symmetric and non-positive matrices too, and it is the most robust of all Krylov type schemes. GMRES, however, becomes expensive both in CPU time and memory requirements if a lot of iterations are needed. In this case BiCGSTAB is preferred, as it requires less memory, and the cost of the iterations is constant.

In some cases the linear system (68) can be simplified before using the iterative schemes. For example, if the diffusion coefficients are zero for some unknowns, they can be solved for by inverting the appropriate submatrix of K. This will result in a new linear equation with fewer unknowns. An example can be electron heat conduction with electron–ion energy exchange. If the ion heat conduction is negligible, the ion energy density can be solved for, and the linear system will become a scalar equation for the electron energy density.

4.6. Fully implicit scheme

In some cases the stiff part of the equations cannot be easily identified, and all variables are involved. An example can be the MHD equations around a strongly magnetized planet (e.g. Jupiter or Saturn), where the strong magnetic field and low density result in a very high Alfvén speed, but the dynamics is dominated by the much slower convective motion and rotation. In this case we have to use a fully implicit scheme to allow larger time steps.

In a fully implicit scheme all the variables and all the terms are handled implicitly, so it can be regarded as a special case of the semi-implicit scheme with $\mathbf{U}_{\rm impl} = \mathbf{U}$, $\mathbf{R}_{\rm impl} = \mathbf{R}$ and $\mathbf{U}^* = \mathbf{U}^n$. The trapezoidal scheme (68) with $\beta = 1/2$ is usually not robust enough for hyperbolic equations, so instead we utilize the second order Backward Difference Formula (BDF2):

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t_n \left[\beta \mathbf{R}(\mathbf{U}^{n+1}) + (1-\beta) \frac{\mathbf{U}^n - \mathbf{U}^{n-1}}{\Delta t_{n-1}} \right], \tag{70}$$

where $\beta = (\Delta t_n + \Delta t_{n-1})/(2\Delta t_n + \Delta t_{n-1})$ [88,19]. One could solve this non-linear system of equations by employing a full Newton iteration, but in all of our applications we found that to be less efficient than solving the linearized form corresponding to the first Newton iteration:

$$P \cdot \left[I - \Delta t_n \beta \frac{\partial \mathbf{R}_1}{\partial \mathbf{U}} \right] \cdot \Delta \mathbf{U} = P \cdot \Delta t_n \left[\beta \mathbf{R}(\mathbf{U}^n) + (1-\beta) \frac{\mathbf{U}^n - \mathbf{U}^{n-1}}{\Delta t_{n-1}} \right], \tag{71}$$

where P is the preconditioner matrix and $\mathbf{R}_1$ is a spatially first order upwind-type discretization in the Jacobian. The use of $\mathbf{R}_1$ helps making the Jacobian matrix diagonally dominant, while using the second order $\mathbf{R}$ on the right hand side still keeps the overall scheme second order accurate.

Explicit calculation of the Jacobian with the complex numerical discretization and boundary conditions is very difficult to do. It is much simpler and more general to use a Jacobian-free evaluation:

$$\left[I - \Delta t_n \beta \frac{\partial \mathbf{R}}{\partial \mathbf{U}} \right] \cdot \Delta \mathbf{U} = \Delta \mathbf{U} - \Delta t_n \beta \frac{\mathbf{R}(\mathbf{U}^n + \epsilon \Delta \mathbf{U}) - \mathbf{R}(\mathbf{U}^n)}{\epsilon} + O(\epsilon), \tag{72}$$

where ϵ is a small number. We also need to use a preconditioner to make the scheme efficient. This requires the calculation of an approximate Jacobian matrix, which is based on the first order local Lax-Friedrichs scheme and it is evaluated with numerical derivatives of the flux and source functions. Additional terms (e.g. for Hall MHD) are added as needed. The BILU type preconditioner is again restricted to each grid block (see [19] for more detail). In each Krylov iteration, the matrix–vector products are calculated with the Jacobian-free method (72) followed by an explicit multiplication with the preconditioner matrix.

We have demonstrated that the fully implicit scheme can produce speed-ups of order 10–20 compared to the explicit scheme [19,43].

4.7. Explicit/implicit scheme

In some applications the PDE is only stiff in a part of the computational domain. For example in the magnetosphere simulations the equations are stiff near the Earth, because the Alfvén speed is order 30,000 km/s, while the actual dynamics happens at much lower speeds, order of $10 - 100$ km/s. Such a stiff system can greatly benefit from an implicit time

integration. Far from the Earth, however, the limiting speed is set by the flow of the solar wind, which also determines the dynamics. Here an explicit time integration makes more sense.

The explicit/implicit scheme [19] provides a hybrid and adaptive approach to efficiently handle the above situation. At the beginning of each time step, we set the global time step Δt based on accuracy and robustness considerations. Then we check each grid block for the CFL stability condition. If the condition is fulfilled, the block is assigned to be an 'explicit block', otherwise it is assigned to be an 'implicit block'. Next the explicit and implicit blocks are load balanced separately. We use the logical array `Unused_B` (see Section 3.2.3) to switch off the implicit blocks temporarily, and advance the explicit blocks from time level n to $n+1$. Then the ghost cells of the implicit blocks are filled in with a message passing. Finally we switch off the explicit blocks, and advance the implicit blocks with the implicit scheme as described in the previous subsection. The explicit/implicit scheme can be 2–5 times faster than the fully implicit scheme, and it also uses substantially less memory (see [19] for more detail).

5. Conclusions

We have been working on space weather modeling, and in general, space physics modeling for more than a decade. We started with a single-purpose although very efficient MHD code BATS-R-US, and we have developed it into a multi-purpose, flexible and rather complex magneto-fluid code. We have also created the Space Weather Modeling Framework that can execute and couple multiple models, including BATS-R-US. During this extensive development we have adapted our algorithms to the various challenges. Instead of creating a multitude of codes, we chose to maintain a single code with a layered and modular software architecture. We are confident that this approach has paid off tremendously. Having a single software base greatly reduces the maintenance work, and it allows using various improvements in multiple applications without multiple implementations.

Developing a complex scientific software requires some level of software engineering. The SWMF with its models consists of about 300,000 lines of Fortran code, and 50,000 lines of scripts and Makefiles. BATS-R-US is about 100,000 lines of Fortran 90 code. We have developed formal coding standards, use version control (with CVS), and do nightly tests on several platforms with different compilers, operating systems and number of processors. The nightly tests allow us to discover bugs and unwanted side effects of new features early. The version control software allows multiple developers to contribute to the same code base. It also allows recovering previous versions of the code in case something went wrong and it was not discovered by the nightly tests. The coding standards provide uniformity of the coding style so that the code remains readable and maintainable for the ever changing set of developers and users. While software engineering is hardly ever discussed in this journal, it is a crucial part of large scale scientific software development.

Although the SWMF and BATS-R-US have reached considerable successes, and they are currently used for space weather modeling as well as short-term forecasting [22], it is still not possible to provide accurate long-term prediction of space weather. As we are constantly working on the improvement and development of the SWMF and BATS-R-US, it is our hope that reliable long-term physics-based space weather forecasting will become reality in the not-so-distant future.

Acknowledgments

The heliospheric components of the SWMF are developed as part of the Comprehensive Corona and Heliospheric Model supported by the NSF ATM 0642309 and NASA NNX07AC16G grants. The magnetospheric components are developed as part of the Community-based Whole Magnetospheric Model supported by NSF ATM-0639336 and AFOSR FA9550-07-1-0434 grants. Extension of BATS-R-US to non-ideal MHD was supported by the NASA Applied Information Systems Research Program grant NNX07AV80G. We also acknowledge support from the NSF Cyber-Enabled Discovery and Innovation grant AGS-1027192. The Block Adaptive Tree Library, the radiative transfer and multi-material physics were developed by the Center for Radiative Shock Hydrodynamics supported by the Department of Energy NNSA under the Predictive Science Academic Alliance Program by grant DEFC52-08NA28616. The General Input Parameter Handling Toolkit development was supported by the DRDA-07-1583 grant from the Jet Propulsion Laboratory. Ward Manchester and Fang Fang acknowledge support by the NSF AGS 1023735 and NASA LWS NNX09AJ78G grants. Merav Opher thanks the support of the NSF CAREER Grant ATM-0747654. We thank the NASA Supercomputer Division at Ames and its helpful staff for making it possible to perform many of the simulations presented in this paper.

Appendix A. General Input Parameter Handling Toolkit – GIPHT

Most scientific software reads input parameters from a simple text file with a fixed format. Documentation is typically incomplete and often out of date. The objective of the General Input Parameter Handling Toolkit (GIPHT) is to handle the input parameters for scientific software in a more user-friendly, better documented and more reliable manner than the current practice. This toolkit makes the SWMF and the ever increasing number of physics models in it more accessible, better documented and better maintained. We believe that other scientific software efforts could also benefit from the GIPHT.

The main concept of GIPHT is to keep the input parameters in a simple, easy-to-read and modify text file, while the possible options for the input parameters are formally described and documented in a separate XML (extended markup language) file. Using the XML description GIPHT can

- provide a GUI to edit the input parameters,
- check the input parameters for correctness,
- produce PDF and/or HTML manuals describing all parameters,
- generate a Fortran 90 code template to read the input parameter file.

The use of the various parts of GIPHT is optional. One can edit the input parameter file directly with any text editor and run the code without using GIPHT at all.

The input parameter file consists of a series of commands followed by parameters that belong to the command and optional comments as shown in Fig. A.16. The commands are recognized from the initial # character. The parameter values and (optional) names are given line by line. The parameter values can be of type integer, real, logical (Boolean) or string. The list of parameters may depend on the values given to previous parameters in the same command, but interdependencies between commands are avoided. There are some special commands, such as `#BEGIN_COMP` and `#END_COMP` that enclose the *section* of input commands belonging to a given component (this is needed by the SWMF only), or the `#RUN` command that indicates the end of the current *session* and the beginning of a new session. When the code reads the `#RUN` command, it executes the simulation with the parameters at that point, and when it reaches a stop condition, it reads in the parameters of the next session. The simulation ends when the last session or the `#END` command is reached. The order of commands within each session is essentially arbitrary. Only commands that change defaults or previous settings have to be put in the file. This keeps the input parameter files at a managable length. It also allows adding new commands without changing the existing input files.

The commands and the parameters are described in XML files. The SWMF and all models that use GIPHT have their own PARAM.XML files. Fig. A.17 shows an example that describes the `#TIMESTEPPING` and `#SCHEME` commands and their parameters. For each command, the first part is a formal description that gives the type, range and default values for the parameters as well as interdependencies among the parameters. This part is used to check the correctness of the input parameter file as well as to generate the pull down menus in the parameter editor. The second part provides an example for the command and its parameters as it appears in the input parameter file, and a manual describing the command, the meaning of the parameters, tips on usage, etc. This part is used to produce the manuals and it is also displayed in the parameter editor. The XML

```
#RUN ###############################################################

#STOP
-1                      MaxIteration
10.0                    tSimulationMax

#BEGIN_COMP GM ------------------------------------------------------

Refine the grid every 30 steps
#AMR
30                      DnRefine
T                       DoAutoRefine
10.                     PercentCoarsen
30.                     PercentRefine
800                     MaxTotalBlocks

Switch to second order scheme
#SCHEME
2                       nOrder
Rusanov                 TypeFlux
minmod                  TypeLimiter

#TIMESTEPPING
2                       nStage
0.8                     CflExpl

#END_COMP GM --------------------------------------------------------

#END ################################################################
```

Fig. A.16. A segment of the input parameter file in the GIPHT format. The final simulation time of the session is given by the `#STOP` command. The other commands select the numerical scheme used by the GM component.

```
<command name="TIMESTEPPING">
  <parameter name="nStage" type="integer" min="1" max="2" default="2" />
  <parameter name="CflExpl" type="real" min="0" max="1" default="0.8" />

#TIMESTEPPING
2                       nStage (1 or 2)
0.80                    CflExpl

The nStage parameter defines the number of stages in the Runge-Kutta scheme.
The CflExpl parameter sets the CFL number for explicit time integration.

Default is 2-stage scheme with CflExpl=0.8
</command>

<command name="SCHEME">
        <parameter name="nOrder" type="integer" min="1" max="2" default="1"/>
        <parameter name="TypeFlux" type="string" input="select" case="upper">
            <option name="RUSANOV" default="T" />
            <option name="HLLE"                />
            <option name="HLLD"                />
            <option name="ROE"                 />
        </parameter>
        <if expr="$nOrder == 2">
                <parameter name="TypeLimiter" type="string" input="select"
                                                            case="lower">
                        <option name="minmod" default="T" />
                        <option name="mc"                 />
                        <option name="koren"              />
                </parameter>
                <parameter name="LimiterBeta" type="real" min="1" max="2"
                        default="1.5" if="$TypeLimiter ne 'minmod"'/>
        </if>
#SCHEME
2                       nOrder (1 or 2)
Rusanov                 TypeFlux
mc                      TypeLimiter ! Only for nOrder=2
1.5                     LimiterBeta ! Only if TypeLimiter is NOT 'minmod'

The nOrder parameter determines the spatial and temporal accuracy of
the scheme. The TypeFlux parameter (Rusanov, HLLE, HLLD, Roe)
defines the numerical flux function. The TypeLimiter (minmod, mc, koren)
selects the slope limiter for second order scheme. The BetaLimiter
parameter (from 1.0 to 2.0) is used by the MC and Koren limiters.

The default is the second order Rusanov scheme with the minmod limiter.
</command>
```

Fig. A.17. A segment of the XML file describing the `#TIMESTEPPING` and `#SCHEME` commands. The actual manual is much more detailed.

description of the #SCHEME command shows examples of conditional statements (for example the slope limiter is only read for second order scheme) and explicit list of input options. The conditional expressions (e.g. $`nOrder` == 2) use Perl syntax that are evaluated by the Perl scripts of GIPHT.

Fig. A.18 shows a snapshot of the parameter editor. The editor runs in a standard web browser that interacts with a mini web server (also part of GIPHT). The mini web server transfers user inputs from the web browser to various Perl scripts that can load, modify and save the input parameter file. The same scripts create the dynamic HTML page from the input parameter file and the XML description that is shown in the browser. For example if the user selects the value 1 from the pull-down menu for the `nOrder` parameter of the `#SCHEME` command, then the Perl scripts immediately regenerate the page and the `TypeLimiter` parameter will disappear. The user can click on the `CHECK` button in the upper-left corner any time to check the correctness of the parameter file. The error messages will be highlighted and shown next to the command that caused the error.

The parameter editor shows the manual for the command being edited. GIPHT also contains scripts and Makefiles that can generate a PDF and/or HTML manual of all the commands described in the XML files. In fact we are dynamically generating the SWMF manual every night and make it accessible through the web. This means that the user manual is as up-to-date as the XML files, which usually are.

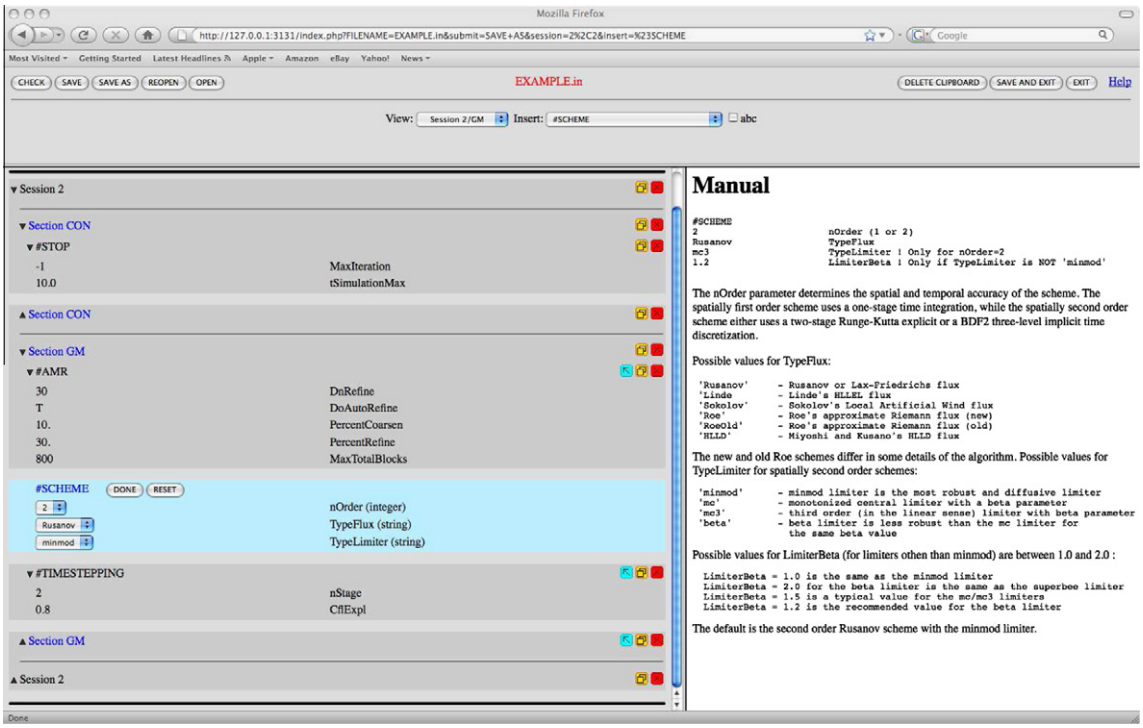

Fig. A.18. Snapshot of the GIPHT parameter editor. The top frame shows the selected session and section (Session 2/GM) of the file, and the command being edited (`#SCHEME`). The frame on the right contains the manual page corresponding to this command. The frame on the left contains the graphical representation of the parameter file. The parameters of the highlighted `#SCHEME` command can be modified by selecting options from the pull down menus. Clicking on the blue, orange, or red icons allow the insertion, copying, or deletion of commands, sections, or sessions, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Finally GIPHT also provides a script that can generate Fortran 90 code that reads in the commands and the parameters. The code is actually very simple, because it uses a module that takes care of reading the input parameter file, distributing it over the processors, finding sessions, sections and commands. Each parameter is read with a single call to the generic `read_var` method that can read all four parameter types (integer, real, logical and string).

We stress that GIPHT is a toolkit and not a highly integrated system. One can use any subset of the tools and modify them as needed. For example one can simply use the Fortran module to read in the input parameter files into a Fortran code. Although there is a script to generate a template Fortran code, it can also be written from scratch. If the XML description of the input parameters exists, one can generate the manuals and/or check the correctness of the input file from the command line. The input files can be edited with any standard text editor or the GIPHT parameter editor, as desired. GIPHT is currently part of the publicly available SWMF, but we may make it accessible as a separate package if there is sufficient interest.

Appendix B. Supercell algorithm

We describe our supercell algorithm for a 2D cylindrical grid where the pole is at $r = 0$. The supercell contains the cells indexed by $i = 1$ and $j = 1 \ldots N_\phi$, and it is surrounded by grid cells indexed with $i = 2$. At the end of each stage of the time stepping scheme we calculate the following quantities:

$$\langle \mathbf{U} \rangle = \frac{1}{N_\phi} \sum_j \mathbf{U}^*_{1,j}, \tag{B.1}$$

$$\langle x^- \mathbf{U} \rangle = \sum_j \max(0, -x_{2,j}) \mathbf{U}^*_{2,j}, \tag{B.2}$$

$$\langle x^+ \mathbf{U} \rangle = \sum_j \max(0, +x_{2,j}) \mathbf{U}^*_{2,j}, \tag{B.3}$$

$$\langle y^- \mathbf{U} \rangle = \sum_j \max(0, -y_{2,j}) \mathbf{U}^*_{2,j}, \tag{B.4}$$

$$\langle y^+ \mathbf{U} \rangle = \sum_j \max(0, +y_{2,j}) \mathbf{U}^*_{2,j}, \tag{B.5}$$

where $\mathbf{U}^*_{i,j}$ is the intermediate solution obtained by the original scheme. In addition, we need the following purely geometrical quantities

$$\langle |x| \rangle = \langle |y| \rangle = \frac{1}{2} \sum_j |x_{2,j}|, \tag{B.6}$$

$$\langle x^2 \rangle = \langle y^2 \rangle = \frac{1}{2} \sum_j x_{2,j}^2, \tag{B.7}$$

where we exploited the rotational and mirror symmetries of the grid around the axis. The 1/2 coefficient is needed because the left and right slopes only use half of the grid cells surrounding the supercell. The summation over the processors requires a single `MPI_allreduce` call. Then we calculate the following least squares type left and right slopes in the x and y directions

$$\mathbf{U}_{x-} = (\langle |x| \rangle \langle U \rangle - \langle x^- U \rangle)/\langle x^2 \rangle, \tag{B.8}$$

$$\mathbf{U}_{x+} = (\langle x^+ U \rangle - \langle |x| \rangle \langle U \rangle)/\langle x^2 \rangle, \tag{B.9}$$

$$\mathbf{U}_{y-} = (\langle |y| \rangle \langle U \rangle - \langle y^- U \rangle)/\langle y^2 \rangle, \tag{B.10}$$

$$\mathbf{U}_{y+} = (\langle y^+ U \rangle - \langle |y| \rangle \langle U \rangle)/\langle y^2 \rangle, \tag{B.11}$$

We use the MC limiter with $\beta = 1.5$ to obtain the limited slopes

$$\overline{\mathbf{U}}_x = \mathrm{minmod}[\beta \mathbf{U}_{x-}, \beta \mathbf{U}_{x+}, (\mathbf{U}_{x-} + \mathbf{U}_{x+})/2], \tag{B.12}$$

$$\overline{\mathbf{U}}_y = \mathrm{minmod}\big[\beta \mathbf{U}_{y-}, \beta \mathbf{U}_{y+}, (\mathbf{U}_{y-} + \mathbf{U}_{y+})/2\big], \tag{B.13}$$

and finally calculate the cell values within the supercell for each j as

$$U^{n+1}_{1,j} = \langle U \rangle + x_{1,j} \overline{\mathbf{U}}_x + y_{1,j} \overline{\mathbf{U}}_y. \tag{B.14}$$

Note that the update is conservative. For smoothly varying U the limiter function takes the least squares slope fit to the cells surrounding the supercell, which makes it second order accurate. We also allow for a supercell with a radius of two ordinary cells. For spherical grids the supercells are applied up to a given radial distance only. The cells inside the supercell are ignored when the time step limit is calculated, and this allows about a factor of 2–3 times larger explicit time steps than the original scheme. The supercell algorithm also allows the solution to propagate across the poles in a smoother fashion. There are much more sophisticated schemes to deal with the pole problem, for example filtering is routinely applied in atmospheric dynamics codes [92].

References

[1] C.C. Goodrich, A.L. Sussman, J.G. Lyon, M.A. Shay, P.A. Cassak, The CISM code coupling strategy, J. Atmos. Sol.-Terr. Phys. 66 (2004) 1469.
[2] J.Y. Lee, A. Sussman, Efficient communication between parallel programs with intercomm, Tech. Rep. 2004-04, UMIACS (2004).
[3] D.L. Brown, W.D. Henshaw, D.J. Quinlan, Overture: an object-oriented framework for solving partial differential equations on overlapping grids, in: SIAM Conference on Object Oriented Methods for Scientific Computing, SIAM, Philadelphia, PA, 1999, pp. 177–184.
[4] SWMF, Space weather modeling framework at <http://csem.engin.umich.edu/swmf>.
[5] G. Tóth, I.V. Sokolov, T.I. Gombosi, D.R. Chesney, C. Clauer, D.L.D. Zeeuw, K.C. Hansen, K.J. Kane, W.B. Manchester, K.G. Powell, A.J. Ridley, I.I. Roussev, Q.F. Stout, O. Volberg, R.A. Wolf, S. Sazykin, A. Chan, B. Yu, J. Kóta, Space weather modeling framework: a new tool for the space science community, J. Geophys. Res. 110 (2005) A12226, doi:10.1029/2005JA011126.
[6] G. Tóth, Flexible, efficient and robust algorithm for parallel execution and coupling of components in a framework, Comput. Phys. Commun. 174 (2006) 793, doi:10.1016/j.cpc.2005.12.017.
[7] C. Hill, C. DeLuca, V. Balaji, M. Suarez, A. da Silva, The ESMF Joint Specification Team, The architecture of the Earth System Modeling Framework, Comput. Sci. Eng. 6 (1) (2004) 18–28, doi:10.1109/MCISE.2004.1255817.
[8] K. Powell, P. Roe, T. Linde, T. Gombosi, D.L. De Zeeuw, A solution-adaptive upwind scheme for ideal magnetohydrodynamics, J. Comput. Phys. 154 (1999) 284–309.
[9] T.I. Gombosi, K.G. Powell, D.L. De Zeeuw, C.R. Clauer, K.C. Hansen, W.B. Manchester, A.J. Ridley, I.I. Roussev, I.V. Sokolov, Q.F. Stout, G. Tóth, Solution-adaptive magnetohydrodynamics for space plasmas: Sun-to-Earth simulations, Comput. Sci. Eng. 06 (2) (2004) 14–35.
[10] J.A. Linker, Z. Mikić, R. Lionello, P. Riley, T. Amari, D. Odstrcil, Flux cancellation and coronal mass ejections, Phys. Plasmas 10 (2003) 1971–1978.
[11] D. Odstrcil, Modeling 3-d solar wind structures, Adv. Space Res. 32 (4) (2003) 497–506.
[12] J. Lyon, J. Fedder, C. Mobarry, The Lyon–Fedder–Mobarry (LFM) global MHD magnetospheric simulation code, J. Atmos. Sol-Terr. Phys. 66 (2004) 1333.
[13] J. Raeder, J. Berchem, M. Ashour-Abdalla, L. Frank, W. Paterson, K. Ackerson, S. Kokubun, T. Yamamoto, J. Slavin, Boundary layer formation in the magnetotail: geotail observations and comparisons with a global MHD simulation, Geophys. Res. Lett. 24 (1997) 951.
[14] P. Janhunen, GUMICS-3: a global ionosphere-magnetosphere coupling simulation with high ionospheric resolution, in: Proceedings of the ESA 1996 Symposium on Environment Modelling for Space-Based Applications, ESA SP-392, 1996, pp. 233–239.
[15] B. van der Holst, R. Keppens, Z. Meliani, A multidimensional grid-adaptive relativistic magnetofluid code, J. Comput. Phys. 179 (2008) 617.
[16] P. MacNiece, K.M. Olson, C. Mobarry, R. deFainchtein, C. Packer, PARAMESH: a parallel adaptive mesh refinement community toolkit, Comput. Phys. Comm. 126 (2000) 330–354.
[17] CHOMBO, Chombo library at <http://seesar.lbl.gov/anag/chombo>.
[18] SAMRAI, Structured adaptive mesh refinement application infrastructure at <https://computation.llnl.gov/casc/samrai>.
[19] G. Tóth, D.L. De Zeeuw, T.I. Gombosi, K.G. Powell, A parallel explicit/implicit time stepping scheme on block-adaptive grids, J. Comput. Phys. 217 (2006) 722–758, doi:10.1016/j.jcp.2006.01.029.
[20] G. Tóth, D.L.D. Zeeuw, T.I. Gombosi, W.B. Manchester, A.J. Ridley, I.V. Sokolov, I.I. Roussev, Sun to thermosphere simulation of the October 28–30, 2003 storm with the space weather modeling framework, Space Weather J. 5 (2007) S06003, doi:10.1029/2006SW000272.
[21] W.B. Manchester, A. Vourlidas, G. Toth, N. Lugaz, I.I. Roussev, I.V. Sokolov, T.I. Gombosi, D.L.D. Zeeuw, M. Opher, Three-dimensional MHD simulation of the 2003 October 28 coronal mass ejection: comparison with LASCO coronagraph observations, Astrophys. J. 684 (2008) 1448–1460.

[22] ISWA, Integrated space weather analysis system at <http://iswa.ccmc.gsfc.nasa.gov>.
[23] A. Pulkkinen, M. Hesse, S. Habib, L.V. der Zel, B. Damsky, F. Policelli, D. Fugate, W. Jacobs, E. Creamer, Solar shield: forecasting and mitigating space weather effects on high-voltage power transmission systems, Nat. Hazards 53 (2009) 333–345, doi:10.1007/s11069-009-9432-x.
[24] CCMC, Community coordinated modeling center at <http://ccmc.gsfc.nasa.gov>.
[25] N.A. Tsyganenko, A magnetospheric magnetic field model with a warped tail current sheet, Planet. Space Sci. 37 (1989) 5.
[26] D. Weimer, A flexible, IMF dependent model of high-latitude electric potential having "space weather" applications, Geophys. Res. Lett. 23 (1996) 2549.
[27] D. Weimer, An improved model of ionospheric electric potentials including substorm perturbations and application to the Geosphace Environment Modeling November 24, 1996, event, J. Geophys. Res. 106 (2001) 407.
[28] S.K. Antiochos, C.R. DeVore, J.A. Klimchuk, A model for solar coronal mass ejections, Astrophys. J. 510 (1999) 485–493.
[29] B. van der Holst, W. Manchester, I. Sokolov, G. Toth, T. Gombosi, D.D. Zeeuw, O. Cohen, Breakout coronal mass ejection or streamer blowout: the bugle effect, Astrophys. J. 693 (2009) 1178–1187.
[30] F. Fang, W.B. Manchester, W.P. Abbett, B. van der Holst, Simulation of flux emergence from the convection zone to the corona, Astrophys. J. 714 (2010) 1649–1657.
[31] W.B. Manchester, T.I. Gombosi, D.L. De Zeeuw, Y. Fan, Eruption of a buoyantly emerging magnetic flux rope, Astrophys. J. 610 (2004) 588.
[32] W. Manchester, T. Gombosi, A. Ridley, I. Roussev, D.D. Zeeuw, I. Sokolov, K. Powell, G. Tóth, Modeling a space weather event from the Sun to the Earth: CME generation and interplanetary propagation, J. Geophys. Res. 109, doi:10.1029/2003JA010150.
[33] N. Lugaz, W.B. Manchester, I.I. Roussev, G. Tóth, T.I. Gombosi, Numerical investigation of the homologous coronal mass ejection events from active region 9236, Astrophys. J. 659 (2007) 788–800.
[34] C. Downs, I.I. Roussev, B. van der Holst, N. Lugaz, I.V. Sokolov, T.I. Gombosi, Toward a realistic thermodynamic magnetohydrodynamic model of the global solar corona, Astrophys. J. 712 (2010) 1219–1231, doi:10.1088/0004-637X/712/2/1219.
[35] B. van der Holst, W. Manchester IV, R. Frazin, A. Vásquez, G. Tóth, T. Gombosi, A date-driven, two-temperature solar wind model with alfvén waves, Astrophys. J. 725 (2010) 1373–1383.
[36] N. Lugaz, W.B. Manchester, T.I. Gombosi, Numerial simulation of the interaction of two coronal mass ejections from Sun to Earth, Astrophys. J. 634 (2005) 651–662.
[37] W.B. Manchester, A.J. Ridley, T.I. Gombosi, D. De Zeeuw, Modeling the Sun–Earth propagation of a very fast CME, Adv. Space Res. 38 (2006) 253–262.
[38] M. Opher, P. Liewer, T. Gombosi, W. Manchester IV, D. De Zeeuw, I. Sokolov, G. Tóth, Probing the edge of the solar system: formation of an unstable jet-sheet, Astrophys. J. L61 (2003) 561.
[39] M. Opher, F.A. Bibi, G. Toth, J.D. Richardson, V.V. Izmodenov, T.I. Gombosi, A strong, highly-tilted interstellar magnetic field near the solar system, Nature 462 (2009) 1036, doi:10.1038/nature08567.
[40] I.V. Sokolov, I.I. Roussev, T.I. Gombosi, M.A. Lee, J. Kóta, T.G. Forbes, W.B. Manchester, J.I. Sakai, A new field line advection model for solar particle acceleration, Astrophys. J. 616 (2004) L171–L174.
[41] J. Kóta, J.R. Jokipii, Transport of CIR accelerated particles, Proc. 26th Int. Cosmic Ray Conf, vol. 6, AIP, 1999, p. 612.
[42] D. De Zeeuw, S. Sazykin, R. Wolf, T. Gombosi, A. Ridley, G. Tóth, Coupling of a global MHD code and an inner magnetosphere model: initial results, J. Geophys. Res. 109 (A12) (2004) A12219, doi:10.1029/2003JA010366.
[43] G. Tóth, Y.J. Ma, T.I. Gombosi, Hall magnetohydrodynamics on block adaptive grids, J. Comput. Phys. 227 (2008) 6967–6984, doi:10.1016/j.jcp.2008.04.010.
[44] A. Glocer, G. Tóth, Y.J. Ma, T. Gombosi, J.-C. Zhang, L.M. Kistler, Multi-fluid BATS-R-US: magnetospheric composition and dynamics during geomagnetic storms, initial results, J. Geophys. Res., in press, doi:10.1029/2009JA014418.
[45] R.A. Wolf, M. Harel, R.W. Spiro, G. Voigt, P.H. Reiff, C.K. Chen, Computer simulation of inner magnetospheric dynamics for the magnetic storm of July 29, 1977, J. Geophys. Res. 87 (1982) 5949–5962.
[46] F. Toffoletto, S. Sazykin, R. Spiro, R. Wolf, Inner magnetospheric modeling with the Rice Convection Model, Space Sci. Rev. 107 (2003) 175–196.
[47] N. Buzulukova, M.-C. Fok, A. Pulkkinen, M. Kuznetsova, T.E. Moore, A. Glocer, P.C. Brandt, G. Toth, L. Rastatter, Dynamics of ring current and electric fields in the inner magnetosphere during disturbed periods: CRCM–BATS-R-US coupled model, J. Geophys. Res. 115 (2010) A05210, doi:10.1029/2009JA014621.
[48] M.W. Liemohn, J.U. Kozyra, C.R. Clauer, A.J. Ridley, Computational analysis of the near-Earth magnetospheric current system during two-phase decay storms, J. Geophys. Res. 106 (2001) 29. 531.
[49] V.K. Jordanova, J.U. Kozyra, G.V. Khazanov, A.F. Nagy, C.E. Rasmussen, M.-C. Fok, A bounce-averaged kinetic model of the ring current ion population, Geophys. Res. Lett. 21 (1994) 2785.
[50] S. Zaharia, V.K. Jordanova, M.F. Thomsen, G.D. Reeves, Self-consistent modeling of magnetic fields and plasmas in the inner magnetosphere: application to a geomagnetic storm, J. Geophys. Res. 111 (2006) A11S14, doi:10.1029/2006JA011619.
[51] M.H. Fok, R.E. Horne, N.P. Meredith, S.A. Glauert, Radiation belt environment model: application to space weather nowcasting, J. Geophys. Res. 113 (2008) A03S08, doi:10.1029/2007JA012558.
[52] A. Glocer, G. Tóth, M. Fok, T. Gombosi, Integration of the radiation belt environment model into the space weather modeling framework, J. Atmos. Sol.-Terr. Phys. 71 (2009) 1653–1663, doi:10.1016/j.jastp.2009.01.003.
[53] A. Glocer, G. Tóth, T.I. Gombosi, D. Welling, Polar Wind Outflow Model (PWOM): modeling ionospheric outflows and their effect on the magnetosphere, initial results, J. Geophys. Res. 114 (2009) A05216, doi:10.1029/2009JA014053.
[54] A. Ridley, T. Gombosi, D. Dezeeuw, Ionospheric control of the magnetosphere: conductance, Ann. Geophys. 22 (2004) 567–584.
[55] A.J. Ridley, Y. Deng, G. Tóth, The global ionosphere thermosphere model, J. Atmos. Sol.-Terr. Phys. 68 (2006) 839–864, doi:10.1016/j.jastp.2006.01.008.
[56] A. Hedin, Extension of the MSIS thermosphere model into the middle and lower atmosphere, J. Geophys. Res. 96 (1991) 1159.
[57] D. Bilitza, International reference ionosphere 2000, Radio Sci. 36 (2001) 261.
[58] S.K. Godunov, Symmetric form of the equations of magnetohydrodynamics (in Russian), Numerical Methods for Mechanics of Continuum Medium, vol. 1, Siberian Branch of USSR Acad. of Sci., Novosibirsk, 1972, pp. 26–34.
[59] J. Brackbill, D. Barnes, The effect of nonzero $\mathbf{V}\cdot\mathbf{B}$ on the numerical solution of the magnetohydrodynamic equations, J. Comput. Phys. 35 (1980) 426–430.
[60] C.R. Evans, J.F. Hawley, Simulation of magnetohydrodynamic flows: a constrained transport method, Astrophys. J. 332 (1988) 659–677.
[61] G. Tóth, P.L. Roe, Divergence- and curl-preserving prolongation and restriction formulae, J. Comput. Phys. 180 (2002) 736–750.
[62] I.V. Sokolov, K.G. Powell, O. Cohen, T.I. Gombosi, Computational magnetohydrodynamics, based on solution of the well-posed riemann problem, in: N.V. Pogorelov, E. Audit, G.P. Zank (Eds.), Numerical Modeling of Space Plasma Flows, Astronomical Society of the Pacific Conference Series, vol. 385, 2008, p. 291.
[63] A. Dedner, F. Kemm, D. Kröner, C. Munz, T. Schnitzer, M. Wesenberg, Hyperbolic divergence cleaning for the MHD equations, J. Comput. Phys. 175 (2003) 645–673.
[64] T. Tanaka, Finite volume TVD scheme on an unstructured grid system for three-diemsnional MHD simulations of inhomogeneous systems including strong background potential field, J. Comput. Phys. 111 (1994) 381–389.
[65] T.I. Gombosi, G. Tóth, D.L. De Zeeuw, K.C. Hansen, K. Kabin, K.G. Powell, Semi-relativistic magnetohydrodynamics and physics-based convergence acceleration, J. Comput. Phys. 177 (2001) 176–205.
[66] F. Rogers, Ionization equilibrium and equation of state in strongly coupled plasmas, Phys. Plasmas 7 (2000) 51.
[67] J.P. Boris, A physically motivated solution of the Alfvén problem, Tech. Rep. NRL Memorandum Report 2167, Naval Research Laboratory, Washington, D.C., 1970.

[68] A.J. Ridley, D.L.D. Zeeuw, T.I. Gombosi, K.C. Hansen, I.V. Sokolov, G. Toth, D.T. Welling, Numerical considerations in simulating the global magnetosphere, Ann. Geophys. 28 (2010) 1589–1614, doi:10.5194/angeo-28-1589-2010.
[69] B. van Leer, Towards the ultimate conservative difference scheme. V.A second-order sequel to Godunov's method, J. Comput. Phys. 32 (1979) 101–136.
[70] B. Koren, A robust upwind discretisation method for advection, diffusion and source terms, in: C. Vreugdenhil, B. Koren (Eds.), Numerical Methods for Advection–Diffusion Problems, Vieweg, Braunschweig, 1993, p. 117.
[71] S.A. Jacques, Momentum and energy transport by waves in the solar atmosphere and solar wind, Astrophys. J. 215 (1977) 942–951.
[72] I.V. Sokolov, I.I. Roussev, M. Skender, T.I. Gombosi, A. Usmanov, Transport equation for MHD turbulence: application to particle acceleration at interplanetary shocks, Astrophys. J. 696 (2009) 261–267, doi:10.1088/0004-637X/696/1/261.
[73] A. Harten, P.D. Lax, B. van Leer, On upstream differencing and Godunov-type schemes for hyperbolic conservation laws, SIAM Rev. 25 (1) (1983) 35–61.
[74] B. Einfeldt, C.D. Munz, P.L. Roe, B. Sjögreen, On Godunov-type methods near low densities, J. Comput. Phys. 92 (1991) 273–295.
[75] J.P. Holloway, D. Bingham, C. Chou, F. Doss, R.P. Drake, B. Fryxell, M. Grosskopf, B. van der Holst, R. McClarren, A. Mukherjee, V. Nair, K.G. Powell, D. Ryu, I. Sokolov, G. Tóth, Z. Zhang, Predictive modeling of a radiative shock system, Reliab. Eng. Syst. Safe., in press.
[76] R.G. McClarren, D. Ryu, R. Drake, M. Grosskopf, D. Bingham, C. Chou, B. Fryxell, B. van der Holst, J.P. Holloway, C.C. Kuranz, B. Mallick, E. Rutter, B. Torralva, A physics, informed emulator for laser-driven radiating shock simulations, Reliab. Eng. Syst. Safe., in press.
[77] A. Harten, High resolution schemes for hyperbolic conservation laws, J. Comput. Phys. 49 (1983) 357–393.
[78] I.V. Sokolov, K.G. Powell, T.I. Gombosi, I.I. Roussev, A TVD principle and conservative TVD schemes for adaptive Cartesian grids, J. Comput. Phys. 220 (2006) 1–5, doi:10.1016/j.jcp.2006.07.021.
[79] V. Rusanov, Calculation of interaction of non-steady shock waves with obstacles, J. Comput. Math. Phys. 1 (1961) 267.
[80] I. Sokolov, E.V. Timofeev, J. Ichi Sakai, K. Takayama, Artificial wind – a new framework to construct simple and efficient upwind shock-capturing schemes, J. Comput. Phys. 181 (2002) 354–393, doi:10.1006/jcph.2002.7130.
[81] T. Miyoshi, K. Kusano, A multi-state HLL approximate riemann solver for ideal magnetohydrodynamics, J. Comput. Phys. 208 (2005) 315–344, doi:10.1016/j.jcp.2005.02.017.
[82] P.L. Roe, Approximate Riemann solvers, parameter vectors, and difference schemes, J. Comput. Phys. 43 (1981) 357–372.
[83] S.K. Godunov, A difference scheme for numerical computation of discontinuous solutions of hydrodynamic equations, Mat. Sb. 47 (3) (1959) 271–306 (in Russian).
[84] Q.F. Stout, D.L. De Zeeuw, T.I. Gombosi, C.P.T. Groth, H.G. Marshall, K.G. Powell, Adaptive blocks: a high-performance data structure, in: Proc. Supercomputing'97, 1997.
[85] G. Tóth, The LASY preprocessor and its application to multidimensional simulations, J. Comput. Phys. 138 (1997) 981–990.
[86] M.J. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, J. Comput. Phys. 82 (1989) 67–84.
[87] H. van der Ven, B. Niemann-Tuitman, A. Veldman, An explicit multi-time-stepping algorithm for aerodynamic flows, J. Comput. Appl. Math. 82 (1997) 423.
[88] R. Keppens, G. Tóth, M.A. Botchev, A. van der Ploeg, Implicit and semi-implicit schemes: algorithms, Int. J. Num. Method Fluids 30 (1999) 335–352.
[89] S.C. Eisenstat, Efficient implementation of a class of preconditioned conjugate gradient methods, SIAM J. Sci. Stat. Comput. 2 (1981) 1–4.
[90] H. van der Vorst, Bi-cgstab: a fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 13 (1992) 631–644.
[91] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear equations, SIAM J. Sci. Stat. Comput. 7 (3) (1986) 856–869.
[92] C. Jablonowski, D.L. Williamson, The pros and cons of diffusion, filters and fixers in atmospheric general circulation models, in: P.H. Lauritzen, C. Jablonowski, M.A. Taylor, R.D. Nair (Eds.), Numerical Techniques for Global Atmospheric Models, Lecture Notes in Computational Science and Engineering, vol. 80, Springer, 2011, pp. 389–504.