IOP Conf. Series: Journal of Physics: Conf. Series **837** (2017) 012017 doi:10.1088/1742-6596/837/1/012017

AMITIS: A 3D GPU-Based Hybrid-PIC Model for Space and Plasma Physics

Shahab Fatemi[1,3], Andrew R. Poppe[1,3], Gregory T. Delory[1,3], William M. Farrell[2,3]

(1) Space Sciences Laboratory, University of California, Berkeley, California, USA, (2) NASA Goddard Space Flight Center (GSFC), Greenbelt, Maryland, USA, (3) Solar System Exploration Research Virtual Institute (SSERVI), NASA Ames Research Center, Moffett Field, California, USA

E-mail: shahab@ssl.berkeley.edu

Abstract. We have developed, for the first time, an advanced modeling infrastructure in space simulations (AMITIS) with an embedded three-dimensional self-consistent grid-based hybrid model of plasma (kinetic ions and fluid electrons) that runs entirely on graphics processing units (GPUs). The model uses NVIDIA GPUs and their associated parallel computing platform, CUDA, developed for general purpose processing on GPUs. The model uses a single CPU-GPU pair, where the CPU transfers data between the system and GPU memory, executes CUDA kernels, and writes simulation outputs on the disk. All computations, including moving particles, calculating macroscopic properties of particles on a grid, and solving hybrid model equations are processed on a single GPU. We explain various computing kernels within AMITIS and compare their performance with an already existing well-tested hybrid model of plasma that runs in parallel using multi-CPU platforms. We show that AMITIS runs ∼10 times faster than the parallel CPU-based hybrid model. We also introduce an implicit solver for computation of Faraday's Equation, resulting in an explicit-implicit scheme for the hybrid model equation. We show that the proposed scheme is stable and accurate. We examine the AMITIS energy conservation and show that the energy is conserved with an error $< 0.2\%$ after 500,000 timesteps, even when a very low number of particles per cell is used.

1. Introduction

Similar to many other fields of science and engineering, the complex nature of the problems encountered in space and plasma physics has demonstrated the need for powerful computer simulations [4, 28]. One computational approach that has been widely used is the kinetic particle-based models. In contrast to magnetohydrodynamics (MHD), kinetic models consider a more detailed analysis of plasma through particle interactions with electromagnetic fields [4]. However, developing a fully kinetic particle-in-cell (PIC) model is computationally intensive, especially for three-dimensional (3D) modeling in space plasma physics and planetary science [57]. In many plasma systems where the ion temporal and spatial scales are of primary interest, hybrid models provide a compromise, and have been successfully applied to a broad range of problems in space and plasma physics [37, 38]. A typical scheme of hybrid models consider ions as kinetic particles, while electrons are a charge neutralizing fluid. Electromagnetic fields are treated in the low-frequency Darwin approximation with the standard assumption of plasma

quasi-neutrality ($n_i \approx n_e$, where n is plasma density and sub-index i and e denote ions and electrons, respectively) [26, 38, 58].

In the last two decades, significant effort has been expended on designing and optimizing high performance hybrid models using multi-CPU parallel programming [7, 29, 43]. However, very few hybrid models exist today that are 3D in both spatial and velocity spaces, fully self-consistent, efficiently parallelized, and have genuine impact and sustained productivity in the space and plasma physics community (e.g., Hybrid-FLASH [29], Halfshell [6], and A.I.K.E.F [43]). These models, similar to any other CPU-based parallel model (e.g., iPIC3D in PIC modeling [39] and BATS-R-US in MHD modeling [23]), require the use of large-scale expensive super-computers, which are not generally affordable and/or readily accessible for every desired simulation. Furthermore, slow communication speeds between individual CPU processors, mainly through networks, introduce significant latencies, limiting the performance of any CPU-based model. Thus, CPU-based models face persistent difficulties in tackling new and/or higher fidelity applications, which limits our ability to make fundamental advances.

Over the last few years, graphics processing units (GPUs) have initiated a revolution in heterogeneous parallel computing, and they are exponentially being used for general purpose applications [33, 44]. The large gap in the performance (shown in Figure 1), availability, capital cost, and power consumption between GPUs and CPUs are the main reasons to migrate from CPU-based towards GPU-based models. The advantages in using GPUs compel us to take a step forward in the field of space and plasma simulation by developing a modeling infrastructure in space simulations (AMITIS) with an embedded three-dimensional (3D in space, velocity, and electromagnetic fields) hybrid plasma solver that runs on GPUs.

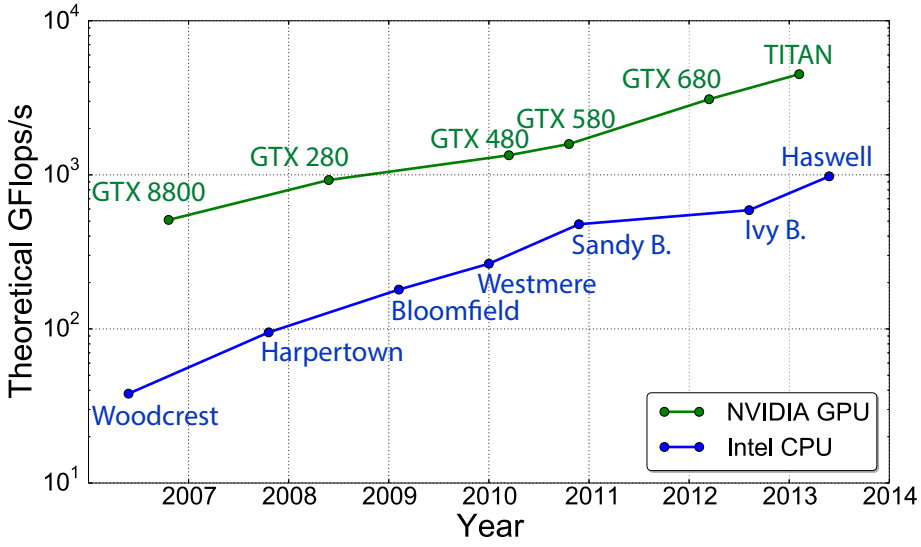

Figure 1. Performance gap between GPUs and CPUs, adopted from www.nvidia.com.

Recently, a few GPU-based models have been developed in plasma physics. However, these models, by the time of this writing, are PIC [1, 8, 12, 15, 16], fluid (MHD) [20, 54], or Vlasov-Boltzmann models [50]. In addition, some of these models are developed in 1D, 2D, and 2.5D [15, 16, 35], and only a few of them are 3D [8, 20]. Here, we present development of the first GPU-based 3D self-consistent hybrid model of plasma for space plasma physics and planetary science, AMITIS, that runs entirely on GPUs. In Section 2 we briefly explain NVIDIA's GPU architecture and compare it with CPUs. Then, in Section 3 we describe hybrid models in detail.

In Section 4 we explain various sections of the AMITIS hybrid model, and in Section 5 provide some of the standard test results for our model and examine its performance, stability, accuracy, and energy conservation.

2. NVIDIA's GPU Architecture

CPUs and GPUs have fundamental differences in their hardware and software architectures. Figure 2 shows a simplified diagram comparing CPU and GPU architectures. A CPU is optimized to enhance the performance of a single thread by using sophisticated control logic while taking advantage of cached memories to reduce any data access latencies [33]. A GPU, in contrast, consists of many cores optimized for parallel tasks, and in total is capable of handling thousands of light-weight threads simultaneously to maximize throughput for parallel programs [33]. Today, a typical CPU with four quad cores can at most run 32 threads concurrently, while a typical NVIDIA GPU runs several thousand active threads concurrently. GPUs also have significantly different memory architecture than CPUs. GPUs contain a hierarchy of progressively smaller, yet faster memory, each of which can serve a particular function in any GPU-based model. The GPU memory types available in NVIDIA devices, listed from the smallest size but the fastest to the largest but slowest, are: registers, shared memory, local memory, constant memory, texture memory, and global memory [11, 55]. In contrast to CPUs, this complex GPU memory hierarchy requires any programmer to engage in sophisticated and intricate memory management in order to achieve high levels of performance in any GPU-based model [9]. Indeed, utilizing memory bandwidth efficiently is critical to algorithm implementation on GPUs [55]. Due to the fundamental differences between GPUs and CPUs, developing a 3D self-consistent GPU-based particle framework and a hybrid plasma field solver requires advanced algorithms and intensive code development, optimization, testing, and diagnostics, completely different from those developed for CPUs.

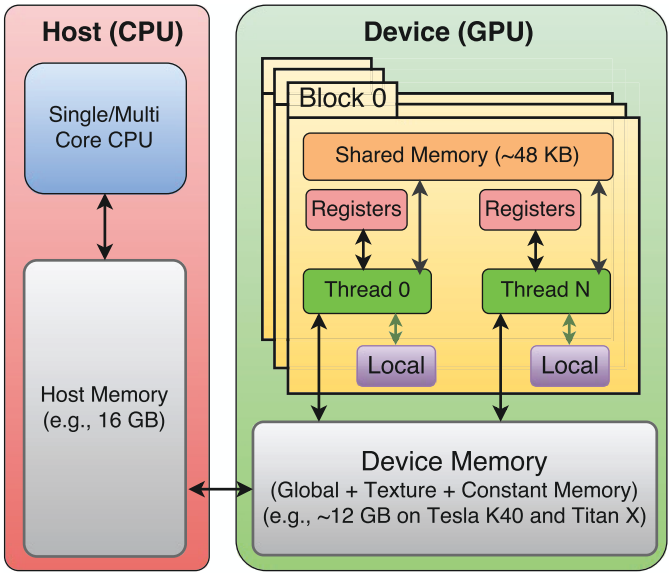

Figure 2. A simplified diagram comparing CPU and GPU architectures.

3. Hybrid Model Description

Hybrid modeling of plasma is a self-consistent kinetic modeling approach that involves solving Maxwell's equations, where a part of plasma is considered as kinetic particles and the rest is

a fluid. Depending on the spatial and temporal scales of problems, various types of hybrid models can be defined [56, 58]. One of the most common types that has a broad range of applications in space and plasma physics considers positively charged ions as kinetic particles whereas electrons are a charge-neutralizing fluid. In this type of hybrid model phenomena occurring at ion inertial and gyroradius scales can be resolved. Examples includes ion-driven plasma waves and instabilities, and plasma interaction with the surface and magnetosphere of objects with sizes on the order of a few ion inertial lengths and the incident ion gyroradius. Figure 3 shows some of the solar system objects covering a broad spectrum of ion length scales. The areas covered with green, blue, and red are suitable for fully kinetic PIC, hybrid, and magnetohydrodynamics (MHD) modeling, respectively. Objects that have a radius (R_{obj}) much smaller than proton gyroradius (r_{gyro}) and much smaller than ion inertial length (δ_i) are mainly suitable for fully kinetic PIC modeling (e.g., Phobos, Deimos, Gaspra, and Itokawa). The objects with radius larger or comparable to the ion gyroradius and the ion inertial length are more suitable for hybrid modeling with kinetic ions and fluid electrons (e.g., Moon, Mars, Callisto, and Rhea). However, if the ion gyroradius and inertial length become too small compared to the size of an object, hybrid models are computationally too expensive and MHD models are more applicable, as long as the kinetic behavior of the ions is not of interest and/or the object does not have a substantial exosphere or ionosphere (which can introduce significantly non-Maxwellian populations unsuitable for a MHD description). We note that defining sharp boundaries between the characteristic scales that each plasma model can resolve is not straightforward; however, the limits, advantages, and disadvantages of any model applied to any problem should be accurately and correctly understood and appreciated.

Generally in PIC and/or hybrid plasma solvers there are a number of basic steps in the calculations that have to be made. These steps are illustrated in Figure 4 and they are as follows [57]:

- **Initialization** where the initial plasma particles and field parameters, simulation domain geometry, and boundary conditions are defined;
- **Solver** where the positions and velocities of all particle species are advanced in time with a relatively small timestep (Δt). Then, macroscopic properties of particles are mapped into grid (if the model is grid-based), fields are solved, and finally, the forces acting on particles are calculated. These steps are the core for every particle solver and are repeated until a final time and/or steady-state solution is achieved; and,
- **Output** where the simulation results are stored in files and the results are analyzed through appropriate diagnostics and visualization tools.

3.1. Hybrid Model Approximations and Equations

Several approximations and assumptions are made in different types of hybrid models. Some of which are rather common in all hybrid models, including our model presented here, and they are all explained in detail in previous literature [10, 26, 37, 38]. Here we only list them without further discussion, but interested readers are encouraged to read the aforementioned references.

- Quasi-neutrality: The total plasma charge density (ρ) is zero.

$$\rho = \rho_e + \rho_I = 0, \tag{1}$$

where ρ_e denotes electron charge densities, and ρ_I is the total ion charge density defined as

$$\rho_I = \sum_{i=1}^{N} \rho_i \tag{2}$$

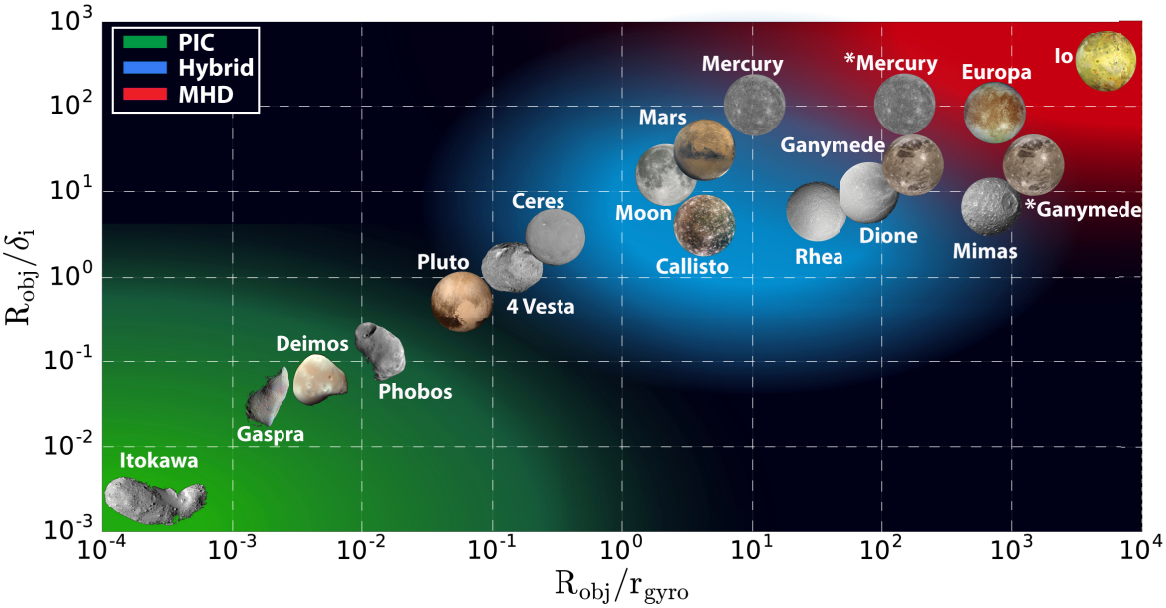

Figure 3. The ratio between the radius of a few solar system objects without any dense atmosphere (R_{obj}) and the typical plasma gyroradius (r_{gyro}) and ion inertial length (δ_i) near those objects. The areas covered with green, blue, and red are suitable for fully kinetic particle-in-cell (PIC), hybrid, and magnetohydrodynamics (MHD) modeling, respectively. The brighter the background, the more feasible those objects are for their associated plasma model. We have included the objects intrinsic magnetic field for those that have an asterisk prior to their name, i.e., Mercury and Ganymede, otherwise only the magnetic field from their surrounding environment has been considered in the ion gyroradius calculation.

where ρ_i denotes ion charge densities, and N is the number of ion species. The quasi-neutrality assumption (Equation 1) removes most electrostatic instabilities and is only valid for grid cells larger than the plasma Debye length λ_D.

- Darwin approximation: The transverse displacement current is neglected from Ampére's law. Thus, Ampére's law is reduced to

$$\mathbf{J} = \frac{\nabla \times \mathbf{B}}{\mu_0}, \tag{3}$$

where $\mathbf{J}$ is the total current density and $\mathbf{B}$ is the magnetic field. Then the electron current density can be written as

$$\mathbf{J}_e = \mathbf{J} - \mathbf{J}_\mathrm{I}, \tag{4}$$

where $\mathbf{J}_\mathrm{I}$ is the total ion current densities defined as

$$\mathbf{J}_\mathrm{I} = \sum_{i=1}^{N} \mathbf{J}_i, \tag{5}$$

where N is the number of ion species.

- Massless electrons ($m_e = 0$): The electron momentum equation with $m_e = 0$ gives an

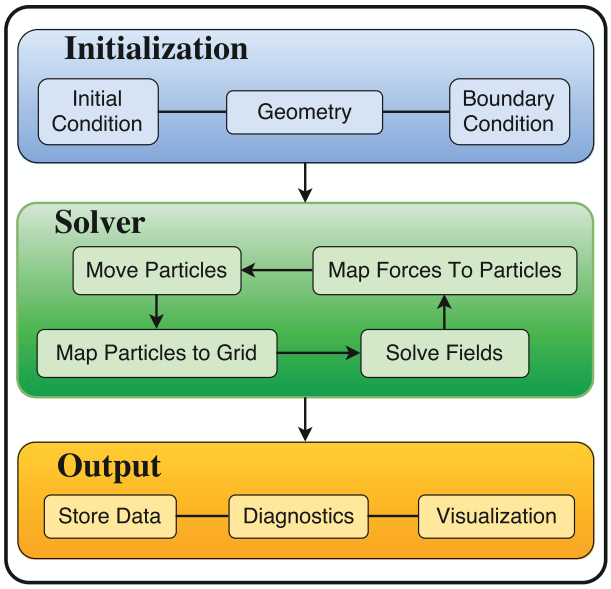

Figure 4. Basic steps in grid-based kinetic plasma simulations inspired from [57].

explicit expression for electric field, $\mathbf{E}$,

$$m_e \frac{d(n_e \mathbf{u}_e)}{dt} = 0 = +\rho_e \mathbf{E} + \mathbf{J}_e \times \mathbf{B} - \nabla \cdot \overline{\mathbf{P}}_e - \rho_e \eta \mathbf{J}, \tag{6}$$

where $\overline{\mathbf{P}}_e$ is the electron pressure tensor and η is resistivity. By replacing $\mathbf{J}$ from Equation 3, $\mathbf{J}_e$ from the expression in Equation 4, and ρ_e from plasma quasi-neutrality (Equation 1), a straightforward solution is obtained for the generalized Ohm's law to calculate the electric field

$$\mathbf{E} = \underbrace{-\frac{\mathbf{J}_\mathrm{I} \times \mathbf{B}}{\rho_\mathrm{I}}}_{\text{convective term}} + \overbrace{(\frac{\nabla \times \mathbf{B}}{\mu_0}) \times \frac{\mathbf{B}}{\rho_\mathrm{I}}}^{\text{Hall term}} - \underbrace{\frac{\nabla \cdot \overline{\mathbf{P}}_e}{\rho_\mathrm{I}}}_{\text{ambipolar term}} + \underbrace{\frac{\eta}{\mu_0} \nabla \times \mathbf{B}}_{\text{Ohmic term}}. \tag{7}$$

The commonly used names for the various electric field terms in Equation 7 are indicated for future reference and ρ_I is defined in Equation 2.

- Electron pressure: Various approaches are taken to treat the electron pressure in Equation 7. As reviewed in [38] some hybrid models assume the electron pressure, and consequently ambipolar electric field term, to be constant or entirely ignored. In some models, the electron pressure is assumed to be isotropic ($\nabla \cdot \overline{\mathbf{P}}_e = \nabla p_e$), where p_e is the scalar electron pressure. Some models determine the electron pressure tensor from the gyrotropic lowest order distributions and reduce it to a diagonal form $\overline{\mathbf{P}}_e = p_\perp (I - \hat{b}\hat{b}) + p_{||} \hat{b}\hat{b}$, where $\hat{b} = \mathbf{B}/|\mathbf{B}|$. A few models also consider the full evolution of the electron pressure tensor derived from Vlasov's equation.
  In the current version of AMITIS, we assume the electron pressure $p_e = n_e k_b T_e$ to be adiabatic, as explained by [29, 19]. The electron pressure is then related to the electron charge density by $p_e \propto |\rho_e|^\gamma$, where γ is the adiabatic index.
- Faraday's law is used to advance the magnetic field in time,

$$\frac{\partial \mathbf{B}}{\partial t} = -\nabla \times \mathbf{E}, \tag{8}$$

where the electric field, **E**, is obtained from Equation 7 and contains no unknowns, thus can be solved straightforwardly. After the electric field is calculated, Faraday's law (Equation 8) is solved to advance the magnetic field, **B**, in time.

The equation of motion for a charged particle with mass m_i and charge q_i at position $\mathbf{r}_i$ and velocity $\mathbf{v}_i$ is

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i, \qquad \frac{d\mathbf{v}_i}{dt} = \frac{q_i}{m_i}(\mathbf{E} + \mathbf{v}_i \times \mathbf{B}), \tag{9}$$

which is an ordinary differential equation.

For this type of hybrid model where the ions are kinetic particles and electrons are a massless charge-neutralizing fluid, modelers are often asked about the treatment of the electron fluid. It is worth mentioning that no flowing "fluid" exists in this type of hybrid models in a way that the fluids are treated in the MHD models, and no continuity equation is required to be explicitly solved for the electron fluid. Hybrid models take advantage of the massless electron fluid concept to calculate the electric field, as shown in Equation 6. However, if so desired, one can estimate the electron current density from Equation 4, and find the electron bulk flow velocity, $\mathbf{u}_e$, through the quasi-neutrality assumption as

$$\mathbf{u}_e = \frac{\mathbf{J}_e}{\rho_e} = \frac{\mathbf{J} - \mathbf{J}_\mathrm{I}}{-\rho_\mathrm{I}}. \tag{10}$$

However, since quasi-neutrality is one of the core assumptions in hybrid models, wherever a positively charged ion exists throughout the simulation domain, a cloud of electrons are assumed to exist around them to maintain quasi-neutrality.

3.2. Hybrid Model Numerical Schemes

Several numerical schemes exist to discretize the hybrid model equations (Equations 7, 8, and 9) and are mostly described in [38], and reviewed by [36, 37, 56, 58]. Two of the most commonly used schemes are Predictor-Corrector (PC) and Current Advance Method and Cyclic Leapfrog (CAM-CL). A detailed description with numerical implementation of the PC scheme can be found in [26, 29], and [56], and CAM-CL is described in [3, 41]. An overview on comparing these two schemes, their advantages and disadvantages, as well as their limitations are explained by [36] and [37].

4. AMITIS Hybrid Model for GPUs

4.1. Model Infrastructure

We developed AMITIS as a 3D, self-consistent hybrid model of plasma that runs on a single CPU-GPU pair. We use the CUDA parallel computing platform and the C/C++ programming language throughout our model. We use NVIDIA GPUs because of their high processing power, high memory bandwidth, low cost and power consumption, and industry support. We use a single CPU as a "host" (shown in Figure 2) to initialize our simulations, call the GPU kernels[1], and store the results into files on the hard drive (initialization and output sections shown in Figure 4). We use a single GPU, termed the "device", for our simulations to perform all computational steps, shown inside the green box in Figure 4, to maximize the model performance.

The NVIDIA GPUs we use for our simulations provide very high throughput and memory bandwidth using single precision operations. Thus, we use single precision floating-point format throughout our model. To minimize round-off errors for single precision operations and to reduce the number of operations, we normalize input parameters to a unit system used in our model.

[1] The parallel portion of an application that is executed on a GPU is called kernel.

All particles and fields are initialized on the host and transferred to the device's global memory. To improve performance during calculations we store all the constant values to the device's constant memory, and we take the advantage of on-chip and fast registers and shared memories, when necessary (see Figure 2 for GPU's memory architecture) [11].

We call the kernels in a sequential order to move the particles and solve the hybrid equations. Since the sequence of every operation per timestep in a plasma solver is crucial, we use synchronization at any level if necessary, and only use asynchronous kernel streams to update grid boundaries and to calculate 3D current densities and fields. We transfer our simulation results from the device to the host memory after every T number of timesteps (often $T \geq 100$), and store them in a Numpy (a Python library) compressed file format. We use Python scripts to visualize our results using Matplotlib library.

Our model is grid-based and we use regular-spaced cell-centered Cartesian grids to solve our equations using finite difference approximations. We do not use adaptive (irregular) grids because of the non-physical forces and instabilities they can introduce into models [22, 53]. We stochastically inject particles into phase space, and we use inversion of cumulative density functions to form distribution functions [48], including Maxwellian, bi-Maxwellian, and Kappa for velocity distributions [40, 46] and uniform, beam-like, and Gaussian for spatial distributions [48].

4.2. Model Sections and Individual Performance Results

Due to the differences between CPU and GPU architectures, several areas of the AMITIS model required intensive research, development, and optimization. In this section, we explain the methods we used in AMITIS to numerically solve every section of the particle model shown in the green box in Figure 4, and benchmark its performance against a highly optimized similar code developed for CPUs. The CPU version of the code used in this section is not a parallel code and only uses a single CPU core. It has been intentionally developed for benchmarking and is highly optimized and well-tested to provide a fair comparison between CPU and GPU performance results. In Section 5, as noted later, we compare AMITIS performance against a well-tested parallel hybrid model than runs on multi-CPU platforms.

For all the simulation results presented in this section we used single precision operations and ran on a machine equipped with an Intel Core i7-4790 3.60GHz CPU, 16 GB DDR3 synchronous 1600 MHz system memory, and a single GeForce GTX TITAN X GPU (from now on called TITAN X) with compute capability 5.0 and 12 GB global memory. We used a single CPU core to run our CPU-based simulations and used a single CPU-GPU pair for AMITIS simulations. We ran only one simulation at a time on our benchmarking machine and made sure that during run-time for each simulation no unnecessary processes were running either on the CPU or on the GPU. We used Ubuntu 14.04 operating system, Linux hybrid 3.13.0-88-generic kernel. We used Intel compiler (icc version 16.0.3) to compile our CPU-based model, and NVIDIA's CUDA compiler (nvcc) to compile AMITIS, unless noted otherwise. Our NVIDIA GPU driver version was 352.68, and our CUDA version was 7.5. Table 1 compares some of the main specifications of Intel Core i7-4790 CPU with TITAN X GPU.

For benchmarking we loaded a 3D simulation box with 64×64×64 cubic-cells of size 200 km with ∼17 million protons as kinetic particles (64 particles per cell), advanced their trajectories, and solved the electromagnetic fields for over 1000 timesteps. All the particles (protons) are singly charged particles with mass 1.0 amu, and they are all uniformly distributed in configuration space with a drifting Maxwellian velocity distribution of bulk speed 350 km/s, thermal speed ∼50 km/s, and background magnetic field 5 nT (typical solar wind parameters near the Earth). We used profilers to estimate runtime simulations and averaged them over 1000 timesteps.

	Intel Core i7-4790	**GeForce TITAN X**
Cores	4	3072
Active threads	8	49152
Core frequency (GHz)	3.60	1.0
Peak performance (GFLOPS/s)	460.8	6144
Peak mem. bandwidth (GB/s)	25.6	336.5
Power consumption TDP (W)	84	250
Power cons. per operation (W/GFLOPS)	0.18	0.04
Current price ($)	300	1000
Price per operation ($/GFLOPS)	0.65	0.16

Table 1. Comparison between Intel Core i7-4790 CPU and GeForce GTX TITAN X GPU. Presented data are taken from http://ark.intel.com and http://www.geforce.com.

4.2.1. Moving Particles Numerous methods exist to numerically solve the charged particles equation of motion (Equation 9) with various levels of accuracy and stability. The most commonly used method to integrate charged particle velocities is the Boris-Buneman method [5]. This method is an explicit scheme with second order accuracy $\mathcal{O}(\Delta t^2)$, has bounded energy error for all timesteps, and conserves phase space volume [34, 49].

We used the Boris-Buneman method to integrate particle velocities and the midpoint method to integrate particle positions with global error of order $\mathcal{O}(\Delta t^2)$. We utilized two different layouts to organize particle data: structure of arrays (SoAs) and array of aligned structures (AoaSs). Generally, many different parallel computer architectures, particularly Single Instruction Multiple Data (SIMD) style like GPUs, prefer SoAs because of its efficient coalesced access to global memory [11, 52]. However, Mei and Tian, 2016 [42] have shown that under some circumstances AoaSs can also provide acceptably high performance compared to SoAs. Figure 5 shows the SoA and AoaS data layouts we have examined in our models. In the AoaS structure we have added two padding elements (foo1 and foo2) to make a 32 bytes data structure and take advantages of aligned memory access. For consistency and to provide a fair comparison between the CPU and GPU versions of our code, we implemented and examined both of these data layouts using different compilers and algorithms. Our benchmarking results are shown in Table 2. We see the impact of compilers (gcc 4.8.4 and intel icc 2016.0.3) as well as the data layouts in the CPU version of our code. We also see that the CPU version of our code runs faster if we advance particle velocities and positions in two "separated" function calls rather than having both "together" in one function call. In contrast, the GPU version of our code runs faster when we advance particles velocity and position "together" in one single function call. This is mainly due to the fundamental differences between the process scheduler on CPUs and GPUs, indicating the importance of finding the most optimized algorithm on each device. Table 2 shows that in general, SoAs provide higher performance compared to AoaSs for both CPU and GPU versions of the particle pushing code, as expected. We also highlighted the fastest performance we achieved running our code on a single CPU core (5.43 ns/particle) and on GPU (0.27 ns/particle) using 256 threads per block. We see that the GPU version of our code provides a speed up $\sim$20X to solve the equation of motion for charged particles.

The maximum number of threads that can be launched per block on the TITAN X is 1024. We examined various configurations of threads per block including 32, 64, 128, 256, 512, and 1024 threads per block. We did not observe noticeable changes in our performance results using any of these configurations to move the charged particles. We also found these results independent from the initial velocity and thermal speed of the particles. It is also worth noting that if we did

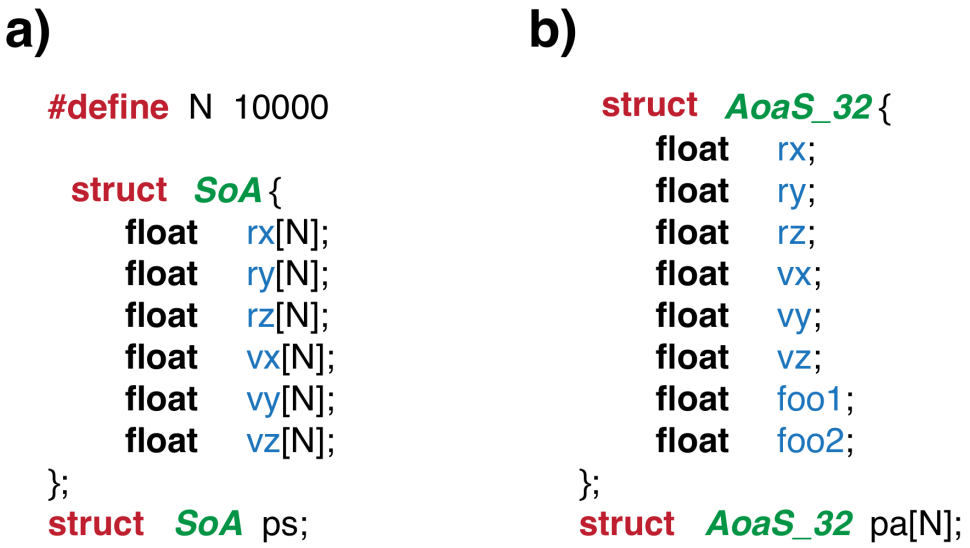

Figure 5. C/C++ data layouts for particles. (a) an example of structure of arrays (SoA), (b) an example of an arrays of aligned structure of size 32 bytes.

Device	Compiler	SoAs		AoSs (Aligned)	
		Separated [ns/particle]	Together [ns/particle]	Separated [ns/particle]	Together [ns/particle]
CPU [Intel Core i7-4790]	gcc/g++	34.47	39.09	33.29	33.28
	intel icc	**5.43**	18.53	6.77	7.74
GPU [TITAN X]	nvcc	0.31	**0.27**	0.52	0.42

Table 2. Performance results for pushing particles using CPUs and GPUs for various data structures, compilers, and algorithm developments. Marked numbers in bold are the fastest runtime achieved using CPU and GPU.

not make a fair comparison, for example by comparing our GPU code performance results with those obtained from CPU code compiled with gcc, a speed up over ∼100X could be achieved. Such large speed ups are questionable, if a fair comparison is not provided.

4.2.2. Mapping Particles to Grid In particle-based kinetic models the particles distribution in space and velocity is defined by a distribution function $f_s(\mathbf{x}, \mathbf{v}, t)$ for each species s, where

$$f_s(\mathbf{x}, \mathbf{v}, t) = \sum_{p=1}^{P} S(\mathbf{x}, \mathbf{x}_p(t))\ \delta(\mathbf{v}, \mathbf{v}_p(t)), \tag{11}$$

where P is the number of particles, S is the shape function, and δ is a Dirac delta function. Macroscopic particle quantities (e.g., charge density, ρ_i, and current density $\mathbf{J}_i$) are mapped onto a grid using shape functions, where

$$\rho_i(\mathbf{x}, t) = \sum_{p=1}^{P} q_p S(\mathbf{x}, \mathbf{x}_p(t)), \qquad \mathbf{J}_i(\mathbf{x}, t) = \sum_{p=1}^{P} q_p \mathbf{v}_p S(\mathbf{x}, \mathbf{x}_p(t)), \tag{12}$$

where q_p is the charge of a particle. Two commonly used shape functions are the cloud-in-cell (CIC) and triangular-shaped cloud (TSC), which are shown in Figure 6 [28]. The macroscopic

quantities are used to calculate the electric field (Equation 7) on a grid in hybrid models, and thus must be accurately calculated.

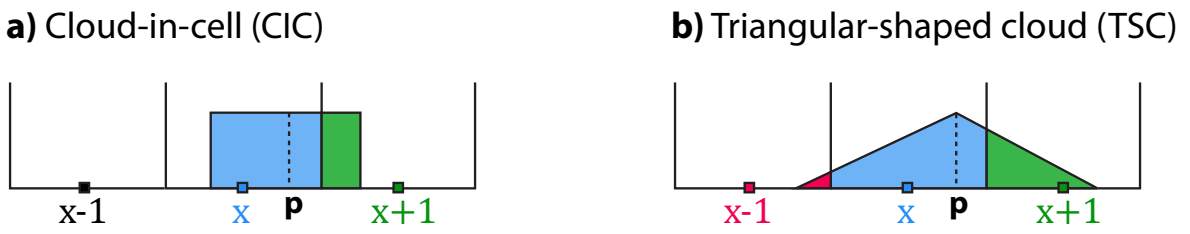

Figure 6. The cloud shape interpolation of charge and current density assignment, taken from [28]. The fraction of charge assignment for a single particle at position p to its neighboring cells using (a) could-in-cell and (b) triangular-shaped cloud shape functions are color-coded here.

Calculating macroscopic quantities associated with particles, and storing these quantities on grid-based models represents one of the largest performance bottleneck operations on GPUs [51]. This is mainly due to race conditions that may arise when several threads attempt to access the same memory address on global or shared memory to store calculated quantities from several particles. In the last decade, several sophisticated and complex algorithms have been proposed to efficiently solve particle mapping and field interpolation for particle algorithms. Examples include clustering a group of grid cells, known as tiling, and particle reordering/sorting [16, 17, 35, 51], and hierarchical Barnes-Hut tree [9]. However, most of these methods have only been developed, optimized, and tested for 1D, 2D and at most 2.5D models [15, 17, 35, 51]. In addition, NVIDIA has included several scalable synchronization mechanisms to its GPUs such as atomic operations. Generally, atomic operations perform mathematical instructions without interference from any other threads. Traditionally, these operations are slow because they may cause serialization for threads in a warp [11]. This has prompted the development of advanced algorithms in the past to avoid race conditions and provide high performance particle mapping to a grid. However, NVIDIA has significantly improved its atomic instructions, especially after the release of CUDA v7.5. From now on,"the atomic functions do not act as memory fences and do not imply synchronization or ordering constraints for memory operations", as stated in the CUDA v7.5 programming guide [14].

The majority of the non-atomic-based algorithms developed to calculate particle macroscopic quantities on a grid ([15, 17, 35]) require particle sorting/reordering and allocations of large memory buffers for particles in order to handle their motion from one grid tile to another. Although this can be minimized, particle sorting adds an extra overhead to model performance. Memory buffers also enforce the use of lower number of particles in a simulation. Since our model is 3D and accommodating more particles per grid cell in the limited global memory of a GPU is crucial, we prefer to use CUDA's atomic operations. Thus, we take the advantage of these operations in AMITIS to calculate particles macroscopic quantities and store them on a grid (mapping particles to grid). Extensive research comparing methods based on atomic and non-atomic operations is required, which is outside the scope of this study.

In AMITIS we have implemented both the CIC and TSC shape functions, and selection of the applied shape function is a free parameter in our model. Due to the larger memory access and complexity of TSC compared to CIC, a lower speed up for TSC is expected compared to CIC. As previously explained, we perform a fair comparison between optimized CPU and GPU versions of our code. Table 3 compares model performance to calculate particle charge density on a regularly spaced 3D grid using various number of grid cells and particles per cell (ppc) using the CIC shape function (averaged over 1000 timesteps). We see a minimum speed up

of 25X with over 30X for large number of particles. This is mainly due to the higher kernel occupancy and cached memory access for larger sized problems on GPUs. Unlike solving the equation of motion, the results here depend on the number of threads per block. For these benchmark experiments we used 256 threads per block. A lower speed up (∼8X-16X) would be obtained using more than 256 threads per block, because of limited resources available on each streaming multiprocessor (SM) on GPUs to perform atomic operations and using registers to calculate particle shape functions on a grid.

Run	Num grid cell	ppc	Total particles	CPU [ms]	GPU [ms]	Speed up
1	32×32×32	32	1048576	14.28	0.57	25.05
2	32×32×32	64	2097152	28.61	1.07	26.73
3	32×32×32	128	4194304	57.13	2.10	27.20
4	64×64×64	32	8388608	132.32	4.23	31.28
5	64×64×64	64	16777216	263.72	8.41	31.35
6	64×64×64	128	33554432	527.03	16.74	31.48

Table 3. Performance results for mapping particles to grids, averaged over 1000 timesteps and presented in milliseconds (ms).

4.2.3. Explicit-Implicit Scheme for Hybrid Field Solver We use the hybrid model solver explained by Holmström, 2010 [29] and solve the hybrid equations using finite differencing scheme in Cartesian coordinates. If the Ohmic term ($\eta\nabla \times \mathbf{B}/\mu_0$) is not included in Equation 7, an explicit finite difference scheme to solve the Faraday's law (Equation 8) will be stable and accurate as long as the Courant-Friedrichs-Lewy (CFL) condition is satisfied for time stepping [29, 36, 41]

$$\Delta t < \frac{\Omega_g}{\sqrt{3}\pi}\left(\frac{\Delta h}{v_A}\right)^2, \tag{13}$$

where Ω_g is the ion gyrofrequency, and v_A is the Alfvén speed. Δh is the length of a grid cell, and theoretically should satisfy $\Delta h \gg \delta_e$, where δ_e is the electron inertial length. For typical solar wind conditions near the Earth with solar wind bulk flow speed $v_{\rm sw} \simeq 400$ km/s, proton gyrofrequency $\Omega_g \simeq 0.5$ rad/s, and Alfvén speed $v_A \simeq 50$ km/s, the simulation cell size should be $\Delta h \gg 2.5$ km. If Δh=100 km, then the CFL condition is satisfied for $\Delta t < 0.35$ s. The timestep also needs to satisfy $\Delta t < \Delta h/v_{\rm sw}$ to ensure that a particle does not cross a grid cell in a single timestep. Thus, for the solar wind condition mentioned above, $\Delta t < 0.25$ s.

Vacuum regions naturally occur in plasmas and present difficulties for hybrid and kinetic models as $\rho_{\rm I}$ becomes zero in Equation 7, in the absence of the Ohmic term. One proposed solution to address this challenge requires including the Ohmic term from Equation 7 [30]. Another method is including the electron inertial term in Equation 7, which requires considering non-zero electron mass ($m_e \neq 0$) [2]. Omelchenko and Karimabadi, 2012 [45] also proposed an asynchronous framework with inhomogeneous timescales to deal with this problem. We follow Holmström, 2013 [30] by using the Ohmic term in Equation 7, which provides the advantage of handling any low-density and vacuum regions in plasma, as well as allowing the definition of conductive/resistive interior structures of objects (e.g., the Moon [31], Europa [32]). The main challenge in using this term in Equation 7 is that in regions where $\rho_{\rm I}$=0 (e.g., plasma wakes and objects interior) Faraday's law (Equation 8) becomes,

$$\frac{\partial \mathbf{B}}{\partial t} = -\nabla \times \left(\frac{\eta}{\mu_0}\nabla \times \mathbf{B}\right), \tag{14}$$

where η=$\eta(\mathbf{x}, t)$ is resistivity in space, $\mathbf{x}$, and time, t. For a spatially constant resistivity, Equation 14 can be given as

$$\frac{\partial \mathbf{B}}{\partial t} = \frac{\eta}{\mu_0}\nabla^2\mathbf{B}, \tag{15}$$

which is a diffusion equation, and is numerically stable for timesteps

$$\Delta t_d < \frac{\mu_0 \Delta h^2}{2\eta}, \tag{16}$$

which is usually much smaller than the CFL condition (Equation 13). For typical solar wind conditions described previously, and for Δh=100 km, the CFL condition is satisfied for $\Delta t < 0.25$ s. However, if $\eta = 10^7\ \Omega \cdot$ m, then the stability for magnetic diffusion (Equation 16) requires Δt_d <6.0×10^{-4} s, which is nearly 400X smaller than the CFL timestep. In order to physically model the interior structure of various solar system objects that are feasible for hybrid modeling, shown in Figure 3, a much larger resistivity is required. For example, lunar crustal resistivity is $\eta \gg 10^7\ \Omega \cdot$ m [24, 31]. Thus, for Δh=100 km, $\Delta t_d \ll$6.0×10^{-4} s. Hybrid modeling of solar wind plasma interaction with the Moon using explicit finite difference schemes with timesteps smaller than 0.01 s is computationally expensive, and for timesteps smaller than 0.001 s is computationally impossible, especially if the electromagnetic response of the interior to a magnetic discontinuity needs to be fully resolved.

To overcome the diffusive restriction on the timestep, we propose an explicit-implicit scheme. We break down the electric field from Equation 7 into two terms: the electric field without the Ohmic term ($\mathbf{E}_{\text{nonOhmic}}$), and the electric field with the Ohmic term ($\mathbf{E}_{\text{Ohmic}}$),

$$\mathbf{E}_{\text{nonOhmic}} = -\frac{\mathbf{J}_i \times \mathbf{B}}{\rho_{\text{I}}} + (\frac{\nabla \times \mathbf{B}}{\mu_0}) \times \frac{\mathbf{B}}{\rho_{\text{I}}} - \frac{\nabla \cdot \overline{\mathbf{P}}_e}{\rho_{\text{I}}}, \tag{17}$$

$$\mathbf{E}_{\text{Ohmic}} = \frac{\eta}{\mu_0}\nabla \times \mathbf{B}, \tag{18}$$

such that

$$\mathbf{E} = \mathbf{E}_{\text{nonOhmic}} + \mathbf{E}_{\text{Ohmic}}. \tag{19}$$

From Faraday's law (Equation 8), we have

$$\frac{\partial \mathbf{B}_{\text{nonOhmic}}}{\partial t} = -\nabla \times \mathbf{E}_{\text{nonOhmic}} \tag{20}$$

and

$$\frac{\partial \mathbf{B}_{\text{Ohmic}}}{\partial t} = -\nabla \times \mathbf{E}_{\text{Ohmic}}. \tag{21}$$

By the distributive properties of vector operations, this also yields

$$\mathbf{B} = \mathbf{B}_{\text{nonOhmic}} + \mathbf{B}_{\text{Ohmic}}. \tag{22}$$

Now we can solve the non-Ohmic term explicitly (at the CFL timestep) while the Ohmic term is solved *implicitly* using an implicit solver (e.g., Crank-Nicholson [13]). The implicit solver allows us to take a single timestep for the Ohmic term as large as the CFL constraint, bypassing the restrictive diffusive timestep. The non-Ohmic and Ohmic solutions are then re-combined (Equations 19 and 22) for the full solution to Maxwell's equations.

In the AMITIS code, we have implemented the scheme presented above to solve our hybrid equations. We explicitly solve Equation 20 using finite difference scheme presented by [29], and use the well-known Crank-Nicholson method [13] to solve Equation 21. The stability and accuracy of this explicit-implicit method is discussed in Section 5.3 and the results are compared with a fully explicit method.

4.2.4. Finite Difference Stencil Calculation of the total current density (Equation 3), electric field (Equation 7), and magnetic field (Equation 8) requires the use of derivatives and vector operations. In AMITIS we implemented two schemes: a three-point and a five-point stencil along each dimension to numerically solve the derivatives in their finite-differenced form and allow the user to select which stencil schema to choose as an input to simulations. The three-point stencil requires one layer of guard cells (also known as ghost cells or halos) to be added in all directions outside the user-defined simulation grids, as shown in Figure 7a. These guard cells are used to establish the boundary conditions and to allow use of the stencil for the outer most cells of the user-defined simulation grids. The five-point stencil requires two layers of guard cells, shown in Figure 7b. The three-point stencil runs faster but at reduced accuracy, while the five-point stencil has improved accuracy but reduced performance. Examples of five-point stencils of two adjacent grid cells in a 3D domain are shown in Figure 7c. In a five-point stencil, the first derivative of a quantity Q at a point x can be approximated as

$$\frac{dQ(x)}{dx} = \frac{Q(x-2h) - 8Q(x-h) + 8Q(x+h) - Q(x+2h)}{12h} + \mathcal{O}(h^4), \tag{23}$$

where h is the distance between two adjacent cells along x.

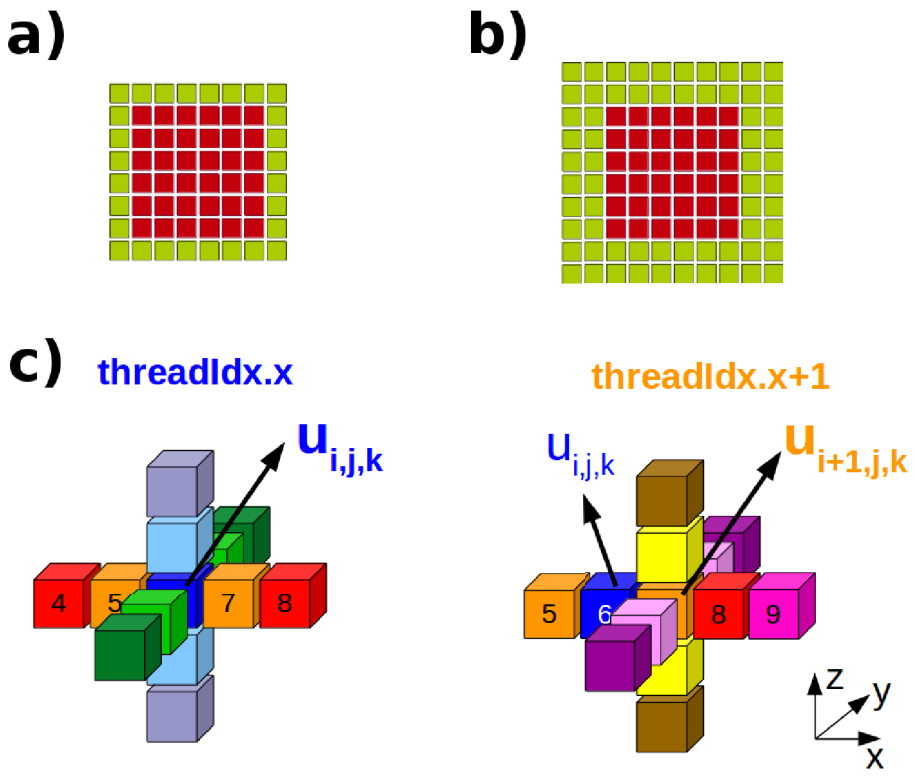

Figure 7. (a) An example of a 6×6 data tile with one layer of guard cells, shown in green, for three-point stencils, (b) an example of a 6×6 data tile with two layers of guard cells for five-point stencils, (c) high order approximations for central based finite difference of a quantity u assigned to a thread of a block.

Since all the grid values are stored in the global memory, reducing the number of reads and writes from global memory and preventing race conditions while calculating derivatives can improve the model performance. Different algorithms have been proposed to minimize global

memory access, which are mainly based on shared memory [11, 55]. Again, the majority of these algorithms have been developed for 1D and 2D domains. Based on our experience in 3D modeling using Maxwell GPU generations and later, we found that the shared memories not only fail to provide considerable improvements in the model performance, but also add a high level of complexity in the model development and programming. In AMITIS we use the global memory to calculate derivatives, but to prevent race conditions we calculate derivatives of every third grid cell in the three-point stencil and every fifth grid cell in the five-point stencil at every call of our computation kernel. Then we repeat the calculation procedure until the derivatives for all grid cells are calculated. Table 4 compares AMITIS performance to calculate a 3D vector curl operation with the CPU version of our code. For a small simulation domain, the GPU code does not perform very well. This is, again, mainly due to the low device occupancy to solve small size problems. However, Table 4 shows a speed up of over 17X can be achieved in calculating three-point stencil curl operations using GPUs compared to CPUs.

Run	Num grid cells	CPU [ms]	GPU [ms]	Speed up
1	16×16×16	0.009	0.019	0.47
2	32×32×32	0.053	0.015	3.53
3	64×64×64	0.436	0.041	10.63
4	128×128×128	4.166	0.234	17.80

Table 4. Performance results to calculate 3D curl operation, averaged over 1000 timesteps.

4.2.5. Mapping Forces to Particles Similar to mapping particles to a grid, shape functions are used for interpolating electromagnetic fields and other forces (e.g., gravity) to the particles position [4, 28]. These forces are used to advance particles in velocity space by solving the equation of motion (Equation 9). Given a single particle p located at position $\mathbf{x}_p$, the force, $\mathbf{F}$, applied to the particle is given by

$$\mathbf{F}(\mathbf{x}_p) = \sum_c \mathbf{F}(\mathbf{x}_c)\ W(\mathbf{x}_c, \mathbf{x}_p), \tag{24}$$

where c are all the cell vertices sounding the particle, and W is the weight function given by the shape function, S, integrated over the cell volume [27]

$$W(\mathbf{x}_c, \mathbf{x}_p) = \int_{\mathbf{x}_c - \Delta x/2}^{\mathbf{x}_c + \Delta x/2} S(\mathbf{x}_c, \mathbf{x}_p)\Delta x, \tag{25}$$

where Δx is the grid cell size. Consistency in the shape function used for mapping particles to grid and interpolating fields to particles is necessary [28]. Therefore, we have included two weight functions based on the CIC and TSC in our model, which can be selected as a free input parameter for a simulation. Since the particles are moving together, if we select a subset of particles, there is a high chance that most of the particles need to access the same grid cells for force interpolation. Therefore, in AMITIS we assign every 32 particles to a single thread of a block. Each thread reads its own dedicated particles, reads the electromagnetic fields from the global memory for the neighboring cells, and interpolates forces to the particles position. Each thread needs to access the global memory for all the neighboring cells only once, unless the particles position have a widely spread spatial distribution. Table 5 compares the AMITIS performance with a highly optimized CPU-based code. With our clustering method on GPU, and without using any shared memory, we achieved >150X speed up in force interpolation to the particles. However, these results may vary by changing the particles thermal speed. We have

chosen an ion thermal speed of ∼50 km/s for the results presented in Table 5. A speed up of ∼25 X was achieved for ions with thermal speed ∼200 km/s. A part of this large speed up in because we do not need to store the interpolated fields into the global memory for every particle. Thus, we do not write into the global memory for this procedure. The method proposed above for particle clustering and assigning each cluster to a thread gives rise to load imbalance and thread divergence on GPUs, which has direct impact on the model performance. This problem can be resolved by assigning every single particle to a single thread per block. This requires every thread reads several grid points from global memory, which again impacts model performance. We did not experience noticeable performance changes using either of these methods.

Run	Num grid cell	ppc	Total particles	CPU [ms]	GPU [ms]	Speed up
1	32×32×32	32	1048576	9.42	0.059	159.6
2	32×32×32	64	2097152	16.33	0.108	151.2
3	32×32×32	128	4194304	32.64	0.202	161.5
4	64×64×64	32	8388608	67.82	0.379	178.9
5	64×64×64	64	16777216	133.21	0.729	182.7
6	64×64×64	128	33554432	256.28	1.441	177.8

Table 5. Performance results for mapping electromagnetic forces to particles, averaged over 1000 timesteps.

5. Test Results

In order to examine the accuracy and stability of our GPU based hybrid model (AMITIS), we analyzed a few standard diagnostics which are explained here. We also compare our model performance with a parallel CPU-based hybrid model implemented in the FLASH code (from now on called hybrid-FLASH). FLASH is a modular, parallel multiphysics simulation code developed by the FLASH Center for Computational Science at the University of Chicago [21]. It is mainly written in Fortran90, and it uses message passing interface (MPI) for interprocessor communications. Although FLASH particle and grid modules have fundamental differences compared to AMITIS, it is the only parallel CPU-based hybrid model of plasma that we have access to. Thus, it is worth noting that we cannot provide a fair comparison between AMITIS and hybrid-FLASH because of their fundamental differences.

5.1. Periodic Plasma Box

We first tested a simple periodic plasma box without any obstacle. We made three simulation runs with different configurations, listed in Table 6. We loaded the simulation box with typical solar wind plasma at 1 AU (near the Earth and the Moon) with ion density $n_0 \simeq 7$ cm^{-3}, ion temperature $T_i \simeq 12$ eV, electron temperature $T_e \simeq 15$ eV, plasma bulk flow speed $|\mathrm{u}_0| \simeq 350$ km/s, and magnitude of the magnetic field $|\mathrm{B}_0| \simeq 5$ nT. Consequently, the ion gyrofrequency $\Omega_g \simeq 0.5$ rad/s, ion inertial length $\delta_i \simeq 86$ km, electron inertial length $\delta_e \simeq 2$ km, Alfvén speed $v_A \simeq 41$ km/s, and ion sound speed $c_s \simeq 46$ km/s.

We only included solar wind protons in the simulations ($m_i \sim 1.0$ amu and $q_i \sim 1.0e$). We assumed the solar wind flows along the -x axis and the magnetic field is only along the +y axis, perpendicular to the plasma flow direction. Thus, the convective electric field $\mathbf{E} = -\mathbf{v} \times \mathbf{B}$ is along the +z axis. The simulation domain is entirely filled in with solar wind protons at initialization (t=0 s), and the domain is assumed to be periodic along all directions. Therefore, when a particle moves out of the simulation domain, it comes back into the box through the opposite boundary. No sources for particle injection and/or loss are considered in the plasma box. The simulation results are expected to be stable and should only show small statistical

runs	grid cells	cell size (Δh)	timestep (Δt)	ppc	total particles
#1	48×24×24	$1.16\delta_i = 100$ km	$2\times10^{-3}\Omega_c = 1\times10^{-3}$ s	32	884736
#2	48×24×24	$1.16\delta_i = 100$ km	$2\times10^{-3}\Omega_c = 1\times10^{-3}$ s	4	110592
#3	48×24×24	$0.58\delta_i = 50$ km	$2\times10^{-4}\Omega_c = 1\times10^{-4}$ s	4	110592

Table 6. List of simulation runs to study the performance cost of the periodic plasma box problem.

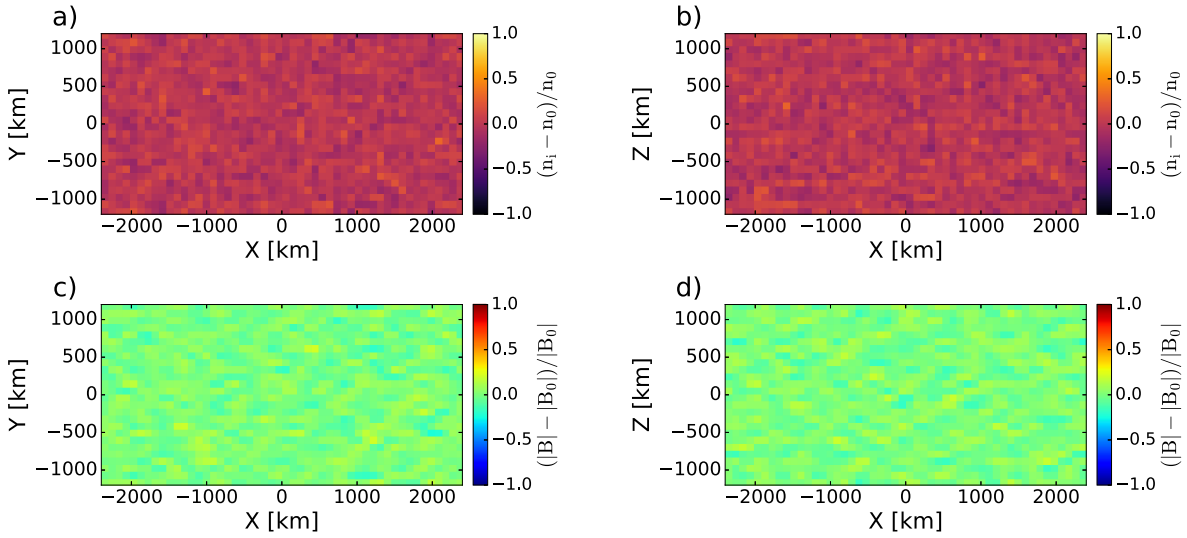

Figure 8. Periodic plasma box simulation results obtained from AMITIS for run #1 in Table 6. (a,b) Plasma number density perturbations relative to the background (initial) ion number density, n_0, (c,d) magnitude of the magnetic field perturbations relative to the background (initial) magnetic field $\mathbf{B}_0$. Panels a and c are cuts in the XY plane at $Z = 0$, viewed from the $+Z$ axis. Panels b and d are cuts in the XZ plane at $Y = 0$, viewed from the $-Y$ axis. In all panels the solar wind plasma flows from right to the left, and the background magnetic field is along the $+Y$ axis and is perpendicular to the plasma flow direction.

fluctuations, depending on hybrid solver scheme, plasma thermal speed, simulation timestep, number of particles per cell, and system round-off/truncation error. We ran each of these experiments for 5×10^5 timesteps, equivalent for solar wind plasma to transit the entire domain for ∼35 times for runs #1 and #2, and for ∼7 times for run #3. We also set η=0 in the ohmic term (Equation 7), thus no implicit solver was used in these experiments.

Figure 8 shows the simulation results for run #1 after 5×10^5 timesteps. The top panels show the variation of the ion number density from the initial solar wind density (n_0) and the bottom panels show the deviation of the magnetic field magnitude ($|\mathbf{B}|$) from the initial magnetic field ($|\mathbf{B}_0|$). We see that the model is stable and does not contain any large fluctuations after 5×10^5 timesteps.

We also calculated the total particle kinetic energy ($\sum_i m_i v_i^2/N_i$, where N_i is the total number of particles), magnetic field energy ($\sum_g B^2/N_g$, where N_g is the total number of grid cells), and electric field energy ($\sum_g E^2/N_g$) at every 200 timesteps, and the results for our three experiments are shown in Figure 9. The energy is conserved throughout all timesteps for our simulation runs with an error of $< 0.2\%$. As shown in Figure 9a, the energy error for run #1 where the simulation cell sizes are slightly larger than ion inertial length ($\sim1.2\delta_i$) and the grid is loaded with moderate number of particles (32 particles per cell) is $< 0.05\%$. In run #2, we intentionally used a smaller

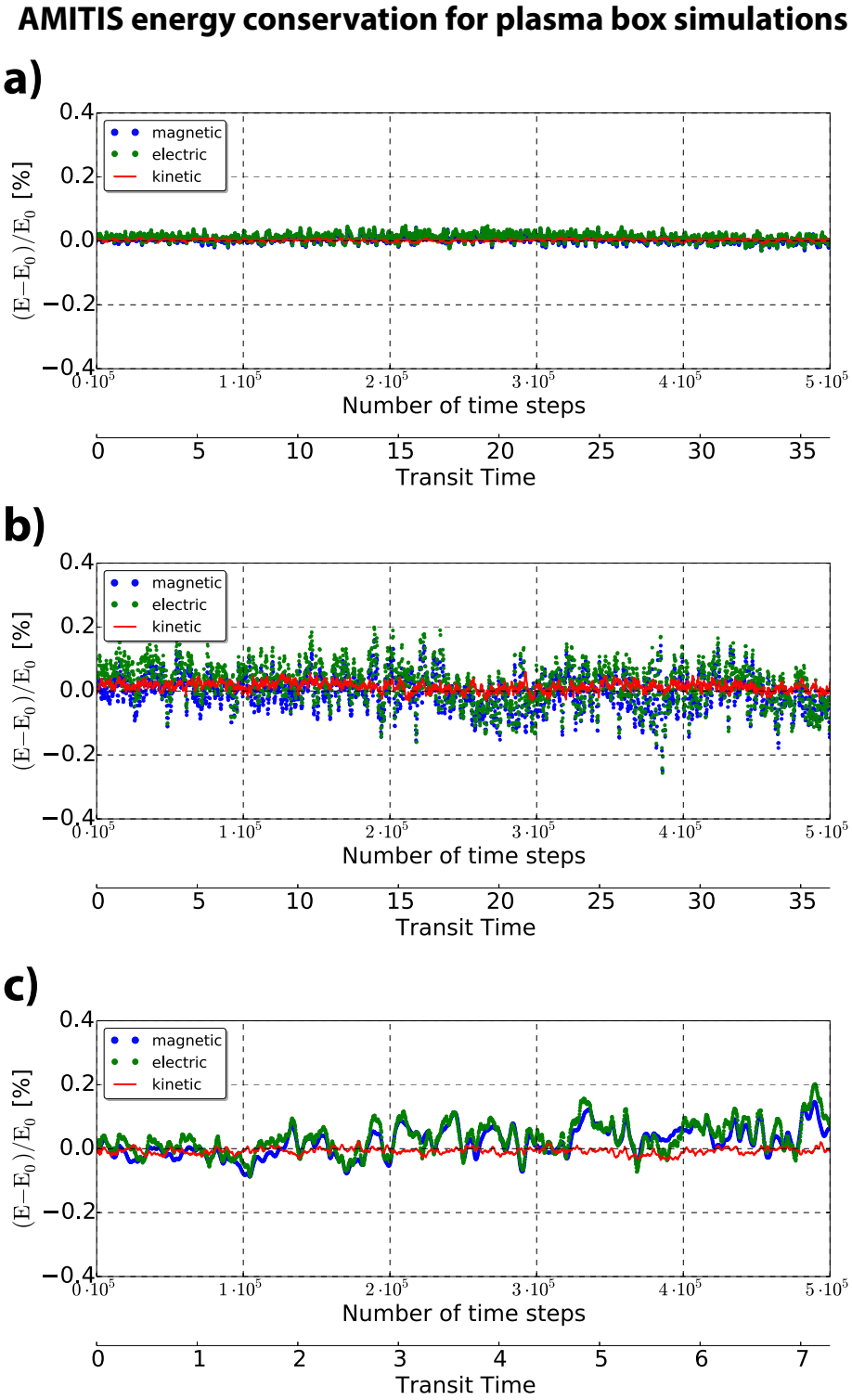

Figure 9. AMITIS energy error $(E - E_0)/E_0$ in percentage, where E_0 is the initial energy, for (a) run #1, (b) run #2, and (c) run #3 listed in Table 6. In all panels, the solid red line indicates particle kinetic energy, blue dots are magnetic field energy, and green dots are electric field energy. The horizontal axes are binned for the number of simulation timesteps and their associated number of transits that the plasma have gone through the entire simulation domain.

number of particles (4 particles per cell) to examine our model stability and energy conservation for fewer number of particles. Figure 9b shows the energy error for run #2 is $\sim 0.2\%$. In run #3, we reduced the simulation cell sizes to $\sim 0.6\delta_i$ and loaded the system with 4 particles per cell, and we see that the energy is conserved with $< 0.2\%$ error, as shown in Figure 9c.

5.2. Scaling Experiment

In order to examine the AMITIS performance and scaling, we provided 9 simulation runs for various number of grid cells and number of particles per cell (ppc), listed in Table 7. We also made identical simulations in FLASH to compare the AMITIS performance against hybrid-FLASH. Again, our simulation domain is periodic along all directions and was loaded with plasma conditions exactly identical to those explained in Section 5.1 (η=0, and no implicit solver is used in AMITIS). We ran the AMITIS code using a single CPU and a single TITAN X GPU. We ran hybrid-FLASH simulations on the Abisko supercomputer[2] using various number of processors, and no GPUs.

run	grid cells	cell size	ppc	total particles
#1	30×30×30	100 km	16	432,000
#2			32	864,000
#3			64	1,728,000
#4	60×60×60	100 km	16	3,456,000
#5			32	6,912,000
#6			64	13,824,000
#7	120×120×120	100 km	16	27,648,000
#8			32	55,296,000
#9			64	110,592,000

Table 7. List of simulation runs for the model scaling experiment.

Figure 10a shows AMITIS execution times for the simulation runs listed in Table 7. As shown in Table 7, the total number of particles for each simulation is increased by a factor of 2 from its previous run. Figure 10a also shows the execution run time for each run takes nearly twice as long compared to the previous run, indicating that AMITIS scales very well as a function of the domain size and the total number of particles. This scaling is very well pronounced for runs #5 to #9, while it is not as good for runs #1 to #4. This is because the total number of particles used in runs #1 to #4 are relatively low for TITAN X to achieve its maximum kernel occupancy, maximum throughput, and maximum memory bandwidth. As soon as the number of particles reach a moderately high value for the TITAN X, a linear scaling as a function of total particle number is achieved.

We ran similar experiments using FLASH by dedicating 48, 96, 384, and 768 CPUs (one processing node has 48 cores), and compared the execution run time per timestep per particle with AMITIS. Results are shown in Figure 10b. We see that depending on the total number of particles and the number of processors, AMITIS runs about 6–40 times faster than hybrid-FLASH.

[2] The Abisko supercomputer is provided by the Swedish National Infrastructure for Computing (SNIC) at the High Performance Computing Center North (HPC2N), Umea University, Sweden. As stated in https://www.hpc2n.umu.se/resources/abisko "the Abisko is comprised of 332 nodes with a total of 15936 CPU cores of which 328 nodes are available to the users. Each node is equipped with 4 AMD Opteron 6238 (Interlagos) 12 core (48 cores per node) 2.6 GHz processors, and 128 GB of DDR3-1600 RAM. The interconnects are Mellanox 4X QSFP 40 Gb/s InfiniBand. The Abisko compute nodes are to be considered complex Non-Uniform Memory Access (NUMA) machines, where the layout of memory in relation to the computing cores used can make a huge difference in real-life performance."

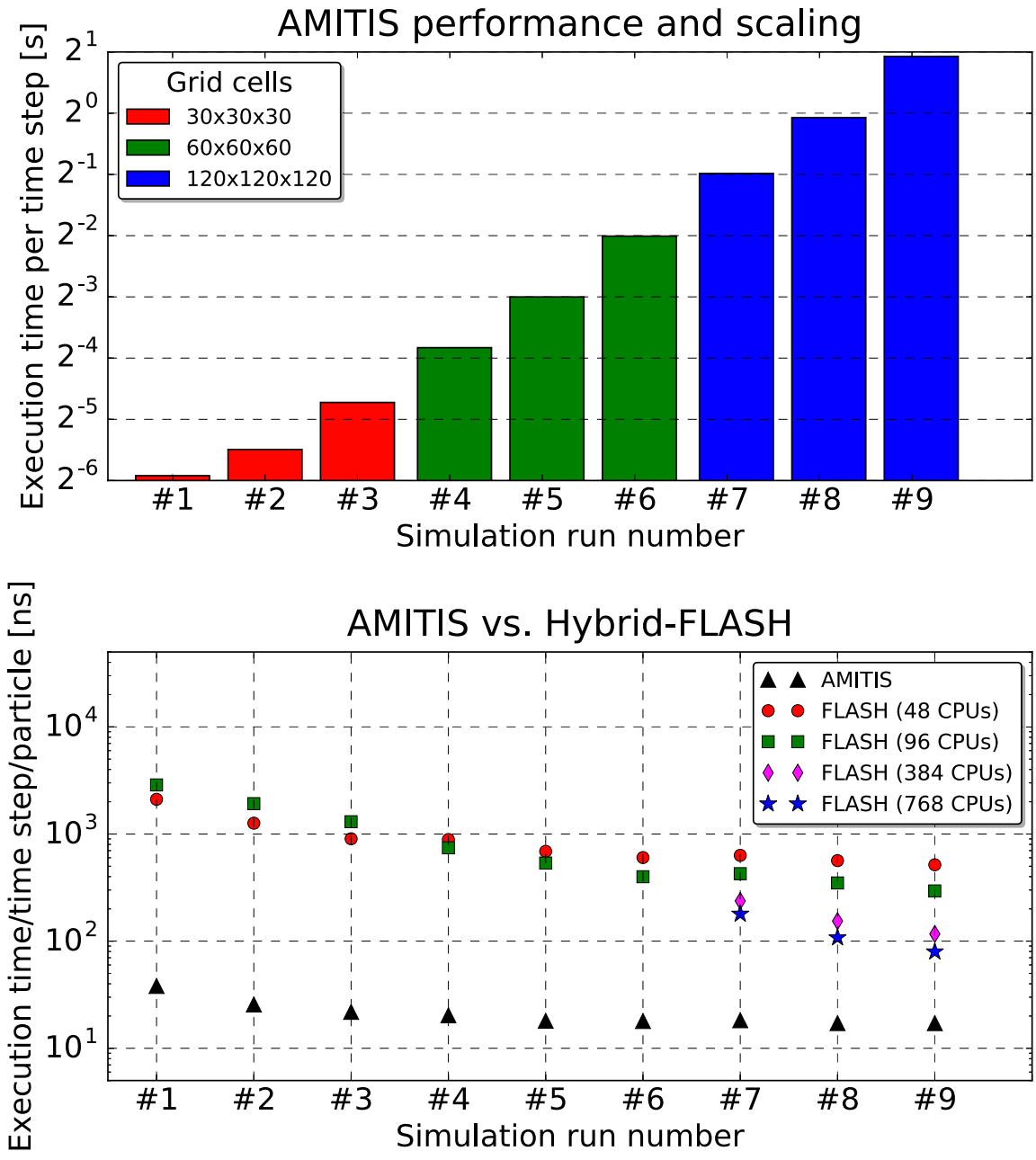

Figure 10. (a) AMITIS performance and scaling for the simulation runs listed in Table 7, (b) Comparison between the AMITIS and hybrid-FLASH execution time. In AMITIS we have used a single CPU-GPU pair, and in hybrid-FLASH we have used different number of CPUs.

5.3. Explicit-Implicit Solver

In order to examine the stability and accuracy of the explicit-implicit scheme discussed in Section 4.2.3, we provided a simulation test for the solar wind plasma interaction with the Moon. Recently, Poppe et al., 2014 [47] compared magnetic field and plasma density observations from the ARTEMIS spacecraft with hybrid simulation results using an explicit solver implemented in the hybrid-FLASH. We took the Poppe et al., 2014 [47] hybrid simulation configurations, listed in Table 8, and applied them into AMITIS. We used a uniform internal resistivity inside the Moon $\eta = 10^7\ \Omega \cdot \mathrm{m}$, identical to that chosen by Poppe et al., 2014 [47], and used the explicit-implicit solver in the AMITIS code.

Parameters	**Values**	**Units**
Solar wind velocity	[-307.0, 0.0, 0.0]	km/s
Solar wind density	10.5	cm^{-3}
Magnetic field	[3.68, 2.23, 0.0]	nT
Ion temperature	43.0	eV
Electron temperature	10.0	eV
Alfvén speed	∼29	km/s
Sound speed	∼69	km/s
Plasma β	∼9	

Table 8. Solar wind plasma parameters during an ARTEMIS spacecraft lunar wake crossing taken from [47].

The Moon, as a first order approximation, can be considered as a non-conducting, atmosphereless body without an intrinsic magnetic field. Thus, the solar wind plasma directly impacts the lunar surface and is absorbed by the Moon. This forms a wake structure downstream and leaves a plasma cavity (vacuum) behind the Moon [25]. Figure 11 provides a global overview on the solar wind plasma interaction with the Moon and compares AMITIS hybrid simulation results using the explicit-implicit solver (Figures 11a and 11b) with hybrid-FLASH explicit solver results presented by [47] (Figures 11c and 11d). Figures 11a and 11c show a density cavity (vacuum) forms downstream behind the Moon, which is due to plasma absorption on the lunar dayside ($X^2 + Y^2 = R_L^2$, where $R_L = 1730$ km is the radius of the Moon). Then, plasma expands into the vacuum region to maintain pressure balance. As the expansion front moves into the vacuum, the density of the ions near the front decreases with distance, forming a rarefied region around the vacuum region. Figures 11b and 11d show the magnitude of the magnetic field in the central lunar wake increases to nearly 4 times higher than the ambient solar wind magnetic field, surrounded by magnetic depressions in the rarefaction region. Magnetic field enhancement and plasma density reduction in the central lunar wake, and magnetic field and plasma density reduction in the rarefaction region are the typical features of the solar wind plasma interaction with the Moon [25]. Since it is impossible to distinguish the differences between the two model results we show the residuals from subtraction of the hybrid-FLASH results from the AMITIS simulations in Figures 11e and 11f. There are minor differences between the two model results which are mainly due to the differences between the explicit-implicit scheme used in the AMITIS and the explicit solver used in the hybrid-FLASH. There are also differences associated with the particle statistics, mapping particles to grids, and mapping forces to particles.

To examine the accuracy of our explicit-implicit solver, in Figure 12 we compared the AMITIS simulation results with hybrid-FLASH simulations and ARTEMIS spacecraft observations presented by [47]. The left panels in Figure 12 compare ARTEMIS spacecraft observations (black lines) with Poppe et al., 2014 [47] explicit hybrid-FLASH simulations (blue lines), and AMITIS explicit-implicit solver results (red lines). We see that both hybrid model simulations have good

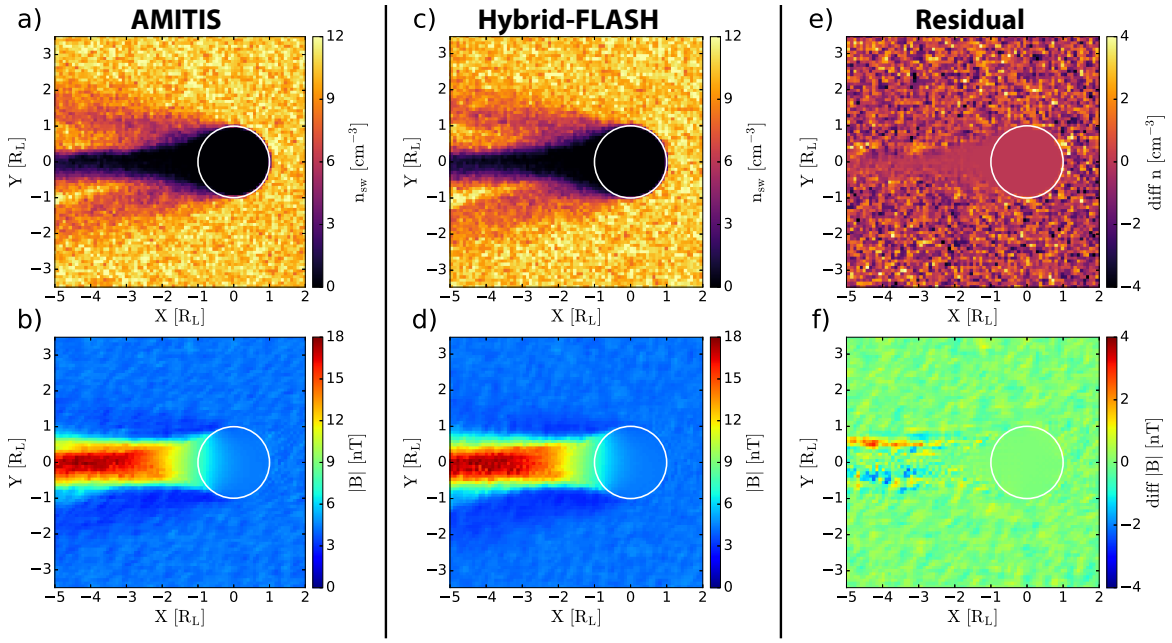

Figure 11. Hybrid simulation results for the moon-solar wind plasma interaction obtained from (a,b) the explicit-implicit solver developed in AMITIS, (c,d) explicit solver developed in hybrid-FLASH, (e,f) residuals from subtraction of the hybrid-FLASH results from AMITIS simulation results. (a,c) Total solar wind plasma density, (b,d) magnitude of the magnetic field. The Moon is located at the center of the coordinate system and is shown by a white circle. The geometry of the cuts is the same as those in Figure 8, and the upstream plasma parameters are listed in Table 8.

agreement with observations. Similar to the results presented in Figure 11 it is impossible to distinguish the differences between the two model results. Thus, in the left panels in Figure 12 we show the residuals from subtraction of the hybrid-FLASH results from the AMITIS simulations on the right panels in Figure 12. The two model results are very similar, but some disagreement between them is expected due to the fundamental differences between the two solvers. This test shows that our explicit-implicit scheme is accurate, and agrees with observations and simulation results obtained from the explicit solver implemented in hybrid-FLASH.

To examine the stability and energy conservation for our explicit-implicit solver, we calculated the particle kinetic energy, magnetic field energy, and electric field energy similar to that in Section 5.1, and the results are shown in Figure 13. At the early stage of the lunar wake development (timesteps $< 2 \times 10^4$), the particle kinetic energy decreases. This is due to the particle absorption on the lunar dayside. As the lunar wake evolves, an ambipolar electric field forms around the lunar wake [18, 25]. This electric field together with particles thermal motion accelerates particles into the wake to fill in the vacuum region shown in Figure 11a. This particle acceleration at later times results in increasing the kinetic energy, shown in Figure 13. After the lunar plasma wake has evolved and reached a steady-state solution, the kinetic energy and electromagnetic energy remain constant with relatively small fluctuations throughout the rest of the simulation. We see from Figure 13 that the explicit-implicit scheme developed for the AMITIS hybrid model is fairly stable and conserves energy with an error $< 0.2\%$.

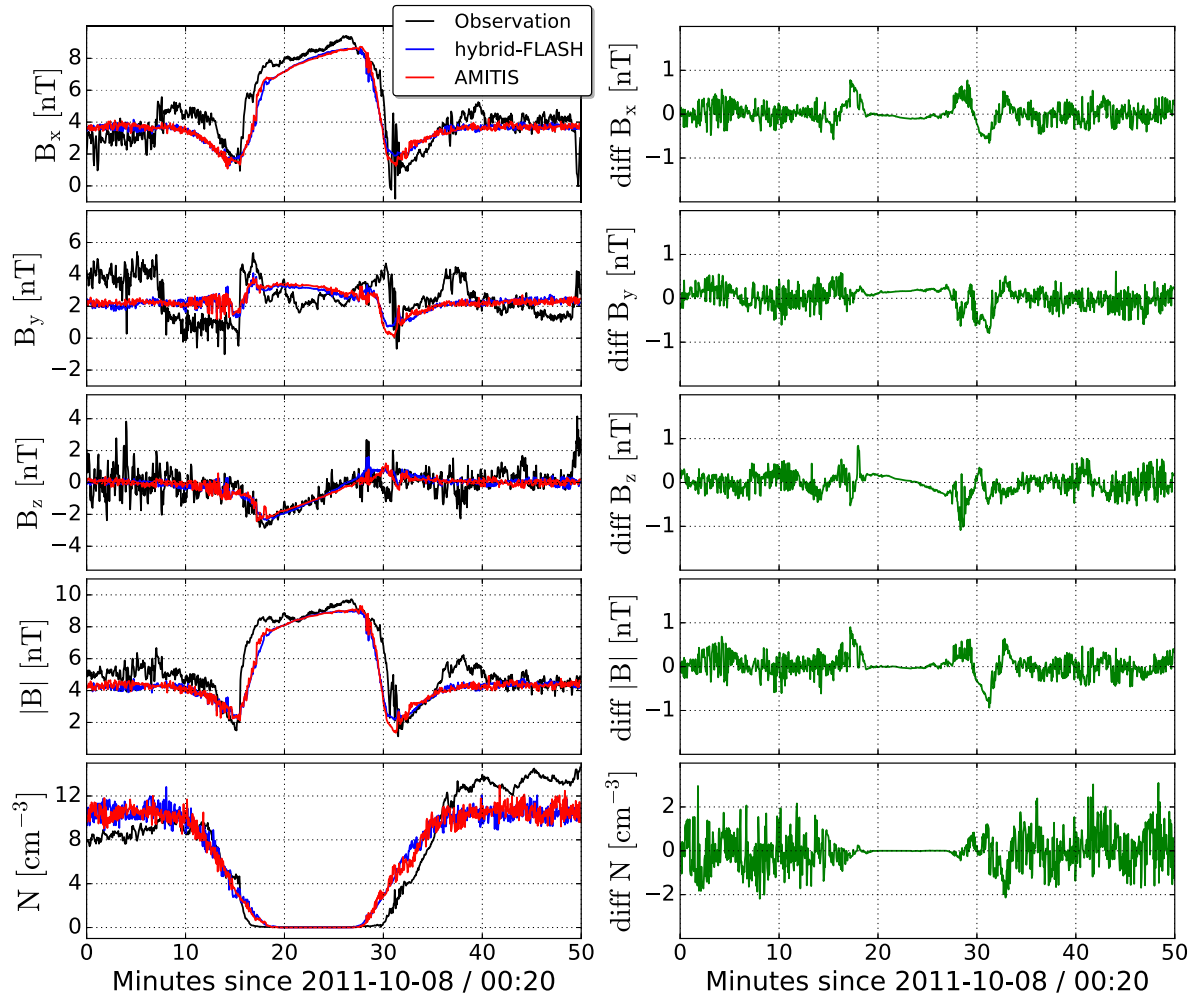

Figure 12. Left panels are a comparison of (black) ARTEMIS data, (blue) hybrid-FLASH using explicit solver, and (red) AMITIS using explicit-implicit solver. Right panels are the magnetic field residuals from subtraction of the hybrid-FLASH results from AMITIS simulation results. Shown from top to the bottom are the three components of the magnetic field, magnetic field magnitude, and plasma density, respectively.

6. Discussion

AMITIS is a GPU-based high performance particle model infrastructure for space simulations. The current version includes a three-dimensional self-consistent hybrid model of plasma and it has the capability to be converted to a full PIC solver by changing its electromagnetic field solver and adding an electron species. The current version of AMITIS runs on a single CPU-GPU pair and on average runs nearly 30X-50X faster than a serial CPU-based model, and ∼5X-10X faster than a parallel CPU-based hybrid model in FLASH. The average cost to buy and maintain a workstation/desktop with a single GPU and a reasonable CPU at the time of this writing is less than ∼3000 US dollar, while a supercomputer with only 128 processors costs over 50,000 USD (see Table 1 for a single CPU vs. single GPU cost comparison). Thus, AMITIS and any other GPU-based models that entirely run on GPUs are more economical compared to parallel CPU-based models. We also showed that AMITIS scales very well with the size of a problem

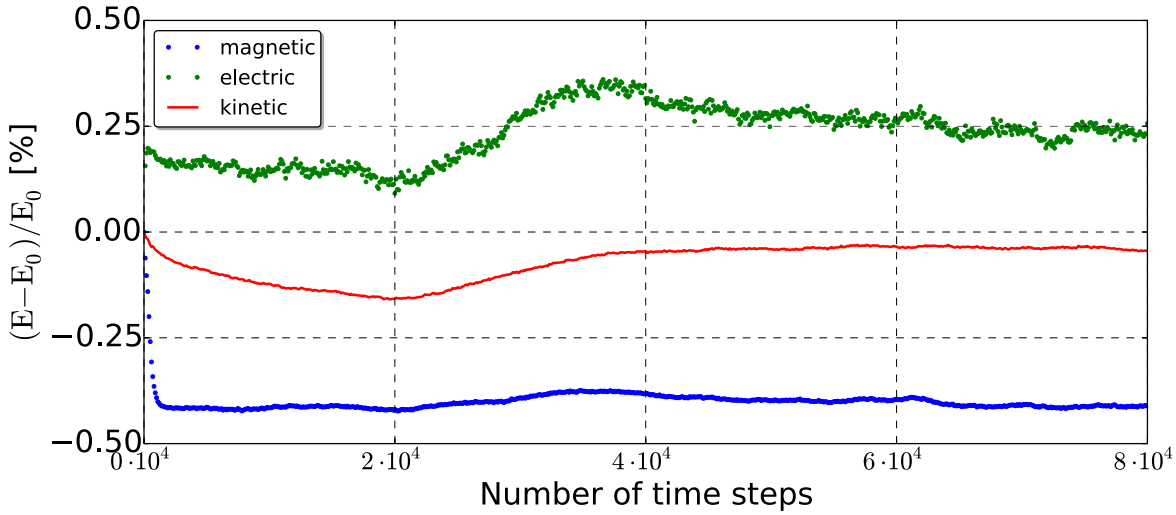

Figure 13. AMITIS explicit-implicit scheme energy error for hybrid simulation of plasma interaction with the Moon, shown in Figure 11. The solid red line indicates particle kinetic energy, blue dots are magnetic field energy, and green dots are electric field energy.

(Figure 10), and conserves energy with an energy error $< 0.2\%$ (Figures 9 and 13).

In the current version of AMITIS the maximum domain size that fits into a 12 GB global memory of the TITAN X GPU is 256×256×256 with 16 particles per cell, or any configuration with similar number of grid cells and particles. This domain size is sufficient to study plasma interactions with different solar system objects that fit within the scales of hybrid modeling shown in Figure 3. However, some plasma physics applications require larger domains (e.g., 1024^3) and/or larger number of particles per cell, which provides motivation to move towards a multi-GPU version of AMITIS in the future.

7. Acknowledgements

Shahab Fatemi is grateful to Mats Holmström for developing a hybrid model of plasma based on FLASH, full and stimulating discussions on hybrid modeling, and sharing his valuable knowledge to the corresponding author. The authors are also grateful to Martin Burtscher at Texas State University for enlightening and fruitful discussions on GPU's memory structure, and CUDA programming. The authors gratefully acknowledge support from NASA's SSERVI institute, grant #NNX14AG16A. We thank NVIDIA's GPU Research Center Program and GPU Technology Conference at San Jose, California for their great support and providing GPUs used in this research. We also acknowledge the DOE NNSA ASC-supported and DOE Office of Science ASCR-supported FLASH Center for Computational Science at the University of Chicago. This research was mainly conducted using GPUs provided by NVIDIA Research Center on computers at Space Sciences Laboratory, University of California at Berkeley, and partly by the Swedish National Infrastructure for Computing (SNIC) at the High Performance Computing Center North (HPC2N), Umeå University, Sweden.

References

[1] Abreu, P., R. A. Fonseca, J. M. Pereira, and L. O. Silva. PIC codes in new processors: A full relativistic PIC code in CUDA-enabled hardware with direct visualization. *IEEE Trans. Plasma Sci.*, 39(2):675–685, 2011.

[2] Amano, T., K. Higashimori, and K. Shirakawa. A robust method for handling low density regions in hybrid simulations for collisionless plasmas. *J. Comput. Phys.*, 275:197–212, 2014.

[3] Bagdonat, T. and U. Motschmann. 3D Hybrid Simulation Code Using Curvilinear Coordinates. *J. Comput. Phys.*, 183(2):470–485, 2002.

[4] Birdsall, C. K. and B. Longdon. *Plasma physics via computer simulation*. McGraw Hill, New York, 1984.

[5] Boris, J. P. Relativistic plasma simulation-optimization of a hybrid code. *Proc, of 4th Conf. on Numerical Simulations of Plasmas*, (Nov.), 1970.

[6] Brecht, S. H. and S. A. Ledvina. The solar wind interaction with the martian ionosphere/atmosphere. *Space Sci. Rev.*, 126(1-4):15–38, 2006.

[7] Brecht, S. H. and S. A. Ledvina. Control of ion loss from mars during solar minimum. *Earth, Planets and Space*, 64(2):165–178, 2012.

[8] Burau, H., R. Widera, W. Honig, G. Juckeland, A. Debus, T. Kluge, U. Schramm, T. E. Cowan, R. Sauerbrey, and M. Bussmann. PIConGPU: A Fully Relativistic Particle-in-Cell Code for a GPU Cluster. *IEEE Trans. Plasma Sci.*, 38(10):2831–2839, 2010.

[9] Burtscher, R. and K. Pingali. An Efficient CUDA Implementation of the Tree-based Barnes Hut N-Body Algorithm. In Wen-Mei, W. H., editor, *GPU Computing Gems Emerald Edition*, chapter 6. Elsevier, 2011.

[10] Byers, J.A, B.I Cohen, W.C Condit, and J.D Hanson. Hybrid simulations of quasineutral phenomena in magnetized plasma. *J. Comput. Phys.*, 27(3):363–396, 1978.

[11] Cheng, J., M. Grossman, and T. McKercher. *Professional CUDA C Programming*. John Wiley & Sons, Inc., 2014.

[12] Claustre, J., B. Chaudhury, G. Fubiani, M. Paulin, and J. P. Boeuf. Particle-In-Cell Monte Carlo Collision Model on GPU-Application to a Low-Temperature Magnetized Plasma. *IEEE Trans. Plasma Sci.*, 41(2): 391–399, 2013.

[13] Crank, J. and P. Nicholson. A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type. *Proc. Camb. Phil. Soc.*, 43(1):50–67, 1947.

[14] CUDA C Programming Guide v7.5, NVIDIA CUDA Toolkit Documentation, . `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`, 2015. Last accessed: Nov 18, 2015.

[15] Decyk, V. K. and T. V. Singh. Adaptable Particle-in-Cell algorithms for graphical processing units. *Comp. Phys. Comm.*, 182(3):641–648, 2011.

[16] Decyk, V. K. and T. V. Singh. Particle-in-Cell algorithms for emerging computer architectures. *Comp. Phys. Comm.*, 185(3):708–719, 2014.

[17] Decyk, V. K., T. V. Singh, and S. A. Friedman. Graphical Processing Unit-based Particle-in-Cell Simulations. In *Proc. Int. Computational Accelerator Physics Conf.*, 2009.

[18] Fatemi, S., M. Holmström, and Y. Futaana. The effects of Lunar surface plasma absorption and solar wind temperature anisotropies on the solar wind proton velocity space distributions in the low-altitude Lunar plasma wake. *J. Geophys. Res.*, 117(A10105):A10105, 2012.

[19] Fatemi, S., M. Holmström, Y. Futaana, S. Barabash, and C. Lue. The lunar wake current systems. *Geophys. Res. Lett.*, 40(1):17–21, 2013.

[20] Feng, X. S., D. K. Zhong, C. Q. Xiang, and Y. Zhang. GPU-accelerated computing of three-dimensional solar wind background. *Sci. China, Earth Sci.*, 56(11):1864–1880, 2013.

[21] FLASH, . Flash Center for Computational Science. University of Chicago and Collaborators. `http://flash.uchicago.edu/site/index.shtml`.

[22] Friedman, A. A second-order implicit particle mover with adjustable damping. *J. Comput. Phys.*, 90(2): 292–312, 1990.

[23] Gombosi, T. I., D. L. Dezeeuw, C. P. T. Groth, K. G. Powell, C. Robert Clauer, and P. Song. *From Sun to Earth: Multiscale MHD Simulations of Space Weather*, pages 169–176. American Geophys. Union, 2003.

[24] Grimm, R. E. and G. T. Delory. Next-generation electromagnetic sounding of the Moon. *Adv. Space Res.*, 50(12):1687–1701, 2012.

[25] Halekas, J. S., D. A. Brain, and M. Holmström. *Moon's Plasma Wake*, pages 149–167. in Magnetotails in the Solar System (eds A. Keiling, C. M. Jackman and P. A. Delamere) John Wiley & Sons, Inc, 2015.

[26] Harned, D. S. Quasineutral hybrid simulation of macroscopic plasma phenomena. *J. Comput. Phys.*, 47 (452):452–462, 1982.

[27] Haugbølle, T., J. T. Frederiksen, and Å Nordlund. photon-plasma: A modern high-order particle-in-cell code. *Physics of Plasmas*, 20(6):062904, 2013.

[28] Hockney, R. W. and J. W. Eastwood. *Computer Simulation Using Particles*. Adam Hilger, Bristol and New

York, 1988.
[29] Holmström, M. Hybrid modeling of plasmas. In *Proceedings of ENUMATH, the 8th European Conference on Numerical Mathematics and Advanced Applications*. Springer, 2010.
[30] Holmström, M. Handling vacuum regions in a hybrid plasma solver. *ASTRONUM-2012, ASP conference series*, (474):202–207, 2013.
[31] Hood, L. L., F. Herbert, and C. P. Sonett. The deep lunar electrical conductivity profile: Structural and thermal inferences. *J. Geophys. Res.*, 87(B7):5311–5326, 1982.
[32] Khurana, K. K., M. G. Kivelson, D. J. Stevenson, G. Schubert, C. T. Russell, R. J. Walker, and C. Polanskey. Induced magnetic fields as evidence for subsurface oceans in Europa and Callisto. *Nature*, 395(6704):777–780, 1998.
[33] Kirk, D. B. and W. W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers, a imprint of Elsevier, 2012.
[34] Knapp, C., A. Kendl, A. Koskela, and A. Ostermann. Splitting methods for time integration of trajectories in combined electric and magnetic fields. *Phys. Rev.*, 92:063310, 2015.
[35] Kong, X., M. C. Huang, C. Ren, and V. K. Decyk. Particle-in-cell simulations with charge-conserving current deposition on graphic processing units. *J. Comput. Phys.*, 230(4):1676–1685, 2011.
[36] Krauss-Varban, D. From Theoretical Foundation to Invaluable Research Tool: Modern Hybrid Simulations. *Proceedings of the 7th International Symposium for Space Simulations (ISSI 7)*, Kyoto University:3, 2005.
[37] Ledvina, S. A., Y.-J. Ma, and E. Kallio. Modeling and Simulating Flowing Plasmas and Related Phenomena. *Space Sci. Rev.*, 139(1-4):143–189, 2008.
[38] Lipatov, A. S., . *The hybrid multiscale simulation technology: an introduction with application to astrophysical and laboratory plasmas*. Springer Science & Business Media, 2002.
[39] Markidis, S., G. Lapenta, and Rizwan-uddin. Multi-scale simulations of plasma with iPIC3D. *Math. and Comp. in Simulation*, 80(7):1509 – 1519, 2010.
[40] Marsch, E. Kinetic physics of the solar wind plasma. In *Physics of the Inner Heliosphere II*, volume 1, chapter 8, pages 45–133. Springer Berlin Heidelberg, 1991.
[41] Matthews, A. Current Advance Method and Cyclic Leapfrog for 2D Multispecies Hybrid Plasma Simulations. *J. Comput. Phys.*, 112(1):102–116, 1994.
[42] Mei, G. and H. Tian. Impact of data layouts on the efficiency of GPU-accelerated IDW interpolation. *SpringerPlus*, 5(1):104, 2016.
[43] Müller, J., S. Simon, U. Motschmann, J. Schüle, K.-H. Glassmeier, and G. J. Pringle. A.I.K.E.F.: Adaptive hybrid model for space plasma simulations. *Comp. Phys. Comm.*, 182(4):946–966, 2011.
[44] Navarro, C. A., N. Hitschfeld-Kahler, and L. Mateu. A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures. *Comm. Comp. Phys.*, 15, 2014.
[45] Omelchenko, Y. A. and H. Karimabadi. HYPERS: A unidimensional asynchronous framework for multiscale hybrid simulations. *J. Comp. Phys.*, 231(4), 2012.
[46] Pierrard, V. and M. Lazar. Kappa distributions: Theory and applications in space plasmas. *Solar Physics*, 267(1), 2010.
[47] Poppe, A. R., S. Fatemi, J. S. Halekas, M. Holmström, and G. T. Delory. ARTEMIS observations of extreme diamagnetic fields in the lunar wake. *Geophys. Res. Lett.*, 41(11):3766–3773, 2014.
[48] Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes, The Art of Scientific Computing*. Cambridge University Press, UK, 3 edition, 2007.
[49] Qin, H., S. Zhang, J. Xiao, J. Liu, Y. Sun, and W. M. Tang. Why is Boris algorithm so good? *Phys. Plasmas*, 20(8), 2013.
[50] Rieke, M., T. Trost, and R. Grauer. Coupled Vlasov and two-fluid codes on GPUs. *J. Comp. Phys.*, 283: 436–452, 2015.
[51] Stantchev, G., W. Dorland, and N. Gumerov. Fast parallel Particle-To-Grid interpolation for plasma PIC simulations on the GPU. *J. Parallel and Distributed Computing*, 68(10):1339–1349, 2008.
[52] Strzodka, R. Abstraction for AoS and SoA Layout in C++. In *GPU Computing Gems Jade Edition*, chapter 31. Addison-Wesley Professional, 2012.
[53] Vay, J.-L., P. Colella, J. W. Kwan, P. McCorquodale, D. B. Serafini, a. Friedman, D. P. Grote, G. Westenskow, J.-C. Adam, a. Heron, and I. Haber. Application of adaptive mesh refinement to particle-in-cell simulations of plasmas and beams. *Phys. Plasmas*, 11(5), 2004.
[54] Wang, P., T. Abel, and R. Kaehler. Adaptive mesh fluid simulations on GPU. *New Astronomy*, 15(7): 581–589, 2010.
[55] Wilt, N. *The CUDA handbook; A comprehensive guide to GPU programming*. Addison Wesley, 2013.
[56] Winske, D. Hybrid Simulation Codes with Application to Shocks and Upstream Waves. *Space Plasma Simulations*, 42:53–66, 1985.
[57] Winske, D. and N. Omidi. A nonspecialist's guide to kinetic simulations of space plasmas. *J. Geophys. Res.*,

101(A8):17287, 1996.
[58] Winske, D. and L. Yin. Hybrid codes: Past, present and future. *Proceedings of ISSS-6*, 6, 2001.