The Astrophysical Journal Supplement Series, 240:8 (12pp), 2019 January https://doi.org/10.3847/1538-4365/aaf1b0

Resistive and Multi-fluid RMHD on Graphics Processing Units

A. J. Wright[1,2] and I. Hawke[1,2]
[1] Mathematical Sciences and STAG Research Centre, University of Southampton SO17 1BJ, UK; A.J.Wright@Soton.ac.uk
[2] Next Generation Computational Modelling Group, University of Southampton SO16 7PP, UK
Received 2018 September 21; revised 2018 November 7; accepted 2018 November 15; published 2019 January 4

Abstract

In this work we present a proof of concept of CUDA-capable, resistive, multi-fluid models of relativistic magnetohydrodynamics (RMHD). Resistive and multi-fluid codes for simulating models of RMHD suffer from stiff source terms, so it is common to implement a set of semi-implicit time integrators to maintain numerical stability. We show, for the first time, that finite volume IMEX schemes for resistive and two-fluid models of RMHD can be accelerated by execution on graphics processing units, significantly reducing the demand set by these kinds of problems. We report parallel speed-ups of over 21× using double-precision floating-point accuracy, and highlight the optimization strategies required for these schemes, and how they differ from ideal RMHD models. The impact of these results is discussed in the context of the next-generation simulations of neutron star mergers.

Key words: magnetohydrodynamics (MHD) – methods: numerical – plasmas – relativistic processes

1. Introduction

Graphics processing units (GPUs) have been steadily gaining attention in the scientific community for a number of years. As the availability of large numbers of GPUs in high-performance computing clusters is growing, and the peak theoretical performance of graphics cards is increasing exponentially, GPUs are being adopted to solve compute-intensive tasks.

GPUs were initially developed for graphics rendering—a highly data-parallel task—and additional hardware was designed to meet these needs using simplified instruction sets, ultimately resulting in slower single thread execution. What graphics cards sacrifice in single thread execution, however, they make up for in sheer numbers of compute cores, allowing massively parallel tasks to be completed rapidly. The introduction of the compute unified device architecture (CUDA; Nickolls et al. 2008) developed by NVIDIA made their GPUs accessible for general purpose programming, and have since seen use in a wide range of scientific applications from neuroimaging (Lee et al. 2012) to deep learning (Gawehn et al. 2018). In this work, we are interested in the potential benefits of GPU execution in evolving different models of relativistic magnetohydrodynamics (RMHD).

Many astrophysical simulations require the evolution of magnetofluids, and by and large these systems are most commonly described by ideal MHD (Kiuchi et al. 2015b; Nouri et al. 2018; Ruiz et al. 2018). Ideal MHD describes a single-fluid, perfectly conducting, charged plasma, and the resulting equations of motion take a simple, balance-law form (Antón et al. 2010). As a result, the standard methods for evolving systems described by ideal MHD are mature, and efficient, and the amount of computation required per time step is relatively low compared to alternative models. Despite this, and while there is still a huge amount of success to be had in modeling events such as neutron star mergers using ideal descriptions (Font 2008; Köppel 2017; Radice 2017), neglecting phenomena such as resistivity may be losing essential physics. The work of Dionysopoulou et al. (2013, 2015) suggests that the dynamics of mergers can be altered significantly by generalizing current models to include finite conductivities. These resistive models of MHD can be taken one step further, as suggested in Andersson et al. (2017a, 2017b, 2017c), and extended to describe multiple interacting charged fluid species, coupled through electromagnetism.

Multi-fluid models of resistive MHD provide a more physically accurate description of these kinds of systems. Multi-fluid effects such as entrainment and the two-stream instability cannot be captured using the single-fluid approximation, and instabilities have been shown to have a major impact in the evolution of neutron star mergers (Price & Rosswog 2006; Obergaulinger et al. 2010; Kiuchi et al. 2015a). Of course, there are difficulties that lie ahead in modeling mergers using these methods. Multi-fluid models inherently contain larger state vectors than single-fluid models, and therefore require additional computation per time step. Furthermore, the multi-fluid description proposed in Andersson et al. (2017c) is not currently in a suitable form for numerical evolution. However, we expect the form of the equations to resemble a balance-law form with the same, potentially very stiff, source term. To prevent numerical instabilities arising from stiff source contributions, it is common to employ implicit time integrators of the type in Pareschi & Russo (2004). These kinds of integrators, while they reduce the total execution time that would be achieved with explicit integrators in many regimes, can make these models too slow for practical application.

If resistive, multi-fluid descriptions of MHD for merger simulations are to be achieved, methods need to be developed that will significantly reduce the computational demands set by these models. Investigations into GPU-implementations of ideal MHD models show promise, and significant performance improvements were reported in Zink (2011) and Wong et al. (2011). Until now, there has been no investigation into the execution of alternative models of MHD on graphics cards.

A multidimensional, general-relativistic formulation of four-fluid MHD, as proposed in Andersson et al. (2017a), would require the evolution of over 30 variables, and a huge amount of computation at each time step when compared to ideal single-fluid models—which themselves require the evolution of less than 10 variables. Fortunately, however, the system of equations is highly parallelizable. The time integration for each computational cell is independent, and the flux methods depend upon only a handful of neighbors, meaning that there is a huge potential for speed-ups using a large number of compute cores.

A major barrier to improving speed-ups using massively parallel processors, however, lies in their memory limitations. While there is the potential to execute a large amount of independent computation in parallel over many thousands of cores, whether the substantially increased size of the multi-fluid system will affect the available speed-up, and whether the limitations on the size of the fastest memory on the GPU will become significant, is unclear. It is the purpose of this work to assess whether a possible avenue to efficient multi-fluid models of RMHD is to utilize GPUs to significantly reduce computation times. To do this we present METHOD, a new open-source (see Section 5) multi-fluid, electro-magnetohydrodynamics code that we use to demonstrate significant performance improvements available for resistive, single and two-fluid models of RMHD by utilizing GPUs.

The rest of the this paper is laid out as follows. In Section 2 we describe the models that are implemented in METHOD, and the numerical methods we use to solve the equations of motion; Section 3 regards how METHOD is designed and the optimization strategies implemented to maximize performance; in Section 4 we validate METHOD via comparisons with exact solutions and convergence tests, and report the parallel speed-up on different GPUs; finally, in Section 5, we end with a discussion of how this work relates to the future of neutron star merger simulations.

2. Models and Numerical Methods

METHOD is a multi-physics, numerical code, implementing three approximations of multi-fluid RMHD. Numerically, plasma dynamics differ from fluid dynamics in the restrictions set by Maxwell's equations, specifically the divergence constraints for the electric and magnetic fields. To satisfy these constraints we use the method of hyperbolic divergence cleaning for each model (Komissarov 2007). It is worth pointing out that Amano (2016) uses the constrained transport method for the two-fluid RMHD model, making the argument that divergence cleaning may not be suitable. We have found that, for the applications in which we use METHOD, divergence cleaning works well, and implementing this method requires very little changes to any existing code.

As we believe that multi-fluid descriptions of MHD will be required for only the smallest scales of a neutron star merger simulation, for the purpose of this work we limit ourselves to local regions of spacetime, and thus to special relativistic descriptions. Therefore, we assume a flat Minkowski geometry (i.e., special relativity), use natural units where the speed of light $c = 1$, and absorb the permittivity and permeability, ϵ_0 and μ_0, into the definition of the electromagnetic fields. We also adopt the Einstein summation convention over repeated indices unless explicitly stated otherwise, and use Latin indices for three spatial dimensions.

2.1. Plasma Models

There are three distinct plasma models implemented in METHOD: ideal RMHD; resistive, single-fluid RMHD; and resistive, two-fluid RMHD. As this paper reports on the benefits of GPU - implementations for resistive RMHD, we will only briefly describe the resistive models, and greater detail can be found in the references. For each model, we assume special relativity, use divergence cleaning to impose Maxwell's constraints, and perform the flux calculations using flux vector splitting (FVS) and an asymptotically third-order WENO3 reconstruction (Section 2.2.2).

Each model requires an equation of state to close the system of equations to relate the hydrodynamics quantities. As is common in the literature, we adopt the Γ-law equation of state, $p = \rho e(\Gamma - 1)$, where Γ is the ratio of specific heats, assumed constant here.

2.1.1. Single-fluid, Resistive RMHD

The formulation of the single-fluid, resistive model can be found in more detail in Komissarov (2007) and Dionysopoulou et al. (2013), along with a motivation for the choice of Ohm's law, which we use throughout this work. The equations of motion take the following form:

$$\partial_t \begin{pmatrix} D \\ S^i \\ \tau \\ B^i \\ E^i \\ \varrho \\ \psi \\ \phi \end{pmatrix} + \partial_k \begin{pmatrix} Dv^k \\ S^{ik} \\ S^k - Dv^k \\ -\epsilon^{ijk}E_j + \delta_i^k \phi \\ \epsilon^{ijk}B_j + \delta_i^k \psi \\ J^k \\ E^k \\ B^k \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -J^i \\ 0 \\ \varrho - \kappa\psi \\ -\kappa\phi \end{pmatrix}, \tag{1}$$

where we relate the conserved vector, $\boldsymbol{q} = (D, S^i, \tau, B^i, E^i, \varrho)$, to the primitive quantities, $\boldsymbol{w} = (\rho, v^i, p, B^i, E^i)$, using

$$\begin{pmatrix} D \\ S^i \\ \tau \\ J^i \\ S^{ij} \end{pmatrix} = \begin{pmatrix} \rho W \\ \rho h W^2 v^i + \epsilon^{ijk}E_j B_k \\ \rho h W^2 - p + \frac{1}{2}(E^2 + B^2) - \rho W \\ \varrho v^i + W\sigma[E^i + \epsilon^{ijk}v_j B_k - (v_k E^k)v^i] \\ \rho h W^2 v^i v^j + [p + \frac{1}{2}(E^2 + B^2)]\delta^{ij} - E^i E^j - B^i B^j \end{pmatrix}, \tag{2}$$

and define the Lorentz factor and specific enthalpy as

$$W = 1/\sqrt{1 - v_i v^i}, \tag{3}$$

$$h = 1 + \frac{\Gamma p}{\rho(\Gamma - 1)}. \tag{4}$$

Throughout this paper, we use $\{\rho, v^i, p, e, E^i, B^i\}$ to represent the rest-mass density (measured in the rest-frame of the fluid), velocity (in the lab frame) in the ith direction, hydrodynamic pressure, specific internal energy, and the electric and magnetic fields in the ith direction, respectively.

The conserved quantities $\{D, S^i, \tau\}$ represent the relativistic mass-energy density, momentum density in the ith direction, and a kinetic energy density-like term, and we make the choice to evolve the charge density, ϱ (initializing it through Gauss' law, $\nabla \cdot E = \varrho$) so as to avoid error accumulation from multiple finite-differencing in the case of discontinuous electric fields. We adopt the same definition of the charge density current as Palenzuela et al. (2009) and Dionysopoulou et al. (2013), with $J^i = \varrho v^i + W\sigma[E^i + \epsilon^{ijk}v^jB^k - (v_kE^k)v^i]$.

Finally, the electric conductivity is given by σ, which in the ideal RMHD limit, $\sigma \to \infty$. This leads to numerical difficulties for systems with high conductivities, as the source terms for the electric fields become large. To maintain stability and reduce the restriction set by the CFL condition, it is common to utilize semi-implicit integration schemes such as those of Section 2.2.1 for stiff systems (Palenzuela et al. 2009; Dionysopoulou et al. 2013, 2015).

The final two equations in (1) are a consequence of the divergence cleaning method (Dedner et al. 2002). In METHOD, we set the timescale $\kappa^{-1} = 1$ for all simulations, and initialize the scalar fields $\phi(t=0) = 0$ and $\psi(t=0) = 0$.

2.1.2. Two-fluid, Resistive RMHD

The resistive two-fluid description closely follows the model presented in Amano (2016) and Balsara et al. (2016). It describes two charged species, denoted by $s \in [p, e]$ for protons/positrons and electrons, respectively, with charge-mass ratios given by $\mu_s = q_s/m_s$ (where there is no sum over species, s).

The equations of motion of the two-fluid system describe how the total, hydrodynamic quantities evolve, and how the same quantities weighted with respect to the charge-mass ratio evolve—the sum weighted with respect to the charge-mass ratio is chosen, as it naturally gives rise to charge conservation and a generalized Ohm's law, as we shall see. It is therefore convenient to define an implied sum over species, in which we assume the total contribution of some variable, U, due to the two charged species is given as $U = \sum_s U_s = U_e + U_p := U_s$. For clarity, the conserved quantity, D, is therefore given by the sum of the contributions, $D = D_e + D_p = \sum_s D_s := D_s = \rho_s W_s$.

In the same form as the single-fluid equations, we get the standard balance-law form,

$$\partial_t \boldsymbol{q} + \partial_j \boldsymbol{f}^j(\boldsymbol{q}) = \boldsymbol{s}(\boldsymbol{q}), \tag{5}$$

in which the conserved vector, $\boldsymbol{q} = \{D, S^i, \tau, \bar{D}, \bar{S}^i, \bar{\tau}, B^i, E^i, \psi, \phi\}$ is related to the primitive quantities through

$$\begin{pmatrix} D \\ S^i \\ \tau \\ \bar{D} \\ \bar{S}^i \\ \bar{\tau} \end{pmatrix} = \begin{pmatrix} \rho_s W_s \\ \rho_s h_s W_s^2 v^i{}_s + \epsilon^{ijk}E_jB_k \\ (\rho_s h_s W_s^2 - p_s) + (E^2 + B^2)/2 - \rho_s W_s \\ \mu_s \rho_s W_s \\ \mu_s \rho_s h_s W_s^2 v^i{}_s \\ \mu_s(\rho_s h_s W_s^2 - p_s - \rho_s W_s) \end{pmatrix}, \tag{6}$$

where we are assuming the sum over fluid species, s, and the primitive quantities have the same interpretation as in the single-fluid case, but specific to a charged species. We present the flux and source vectors in the Appendix.

As in the single-fluid case, the conserved quantities $\{D, S^i, \tau\}$ represent the relativistic mass-energy density, momentum density, and kinetic energy-like density—i.e., the sum of contributions of each fluid species. The barred alternatives, $\{\bar{D}, \bar{S}^i, \bar{\tau}\}$, correspond to the weighted sum with respect to the charge-mass ratio, excluding any contribution from the electromagnetic fields.

A note on the plasma frequency, $\omega_p = \sqrt{\mu_s^2 \rho_s}$. A key factor differentiating single and multi-fluid plasmas is one of charge separation, and the so-called "two-fluid effect," which manifests as oscillations of a fluid's constituent charged species. The degree to which this effect can be seen is dependent upon the spatial resolution of the simulation and the plasma skin depth, $\lambda_p = 1/\omega_p$. That is, we expect to see differences between single-fluid and multi-fluid descriptions when the spatial resolution is smaller than the skin depth, or $\Delta x < \lambda_p$, as will be seen.

2.1.3. Primitive Recovery

An important way in which Newtonian and relativistic MHD differ is in the transformation from the evolved, conserved variables, $\boldsymbol{q}$, and the primitive variables, $\boldsymbol{w}$. In the relativistic regime, there is no closed form for the primitive quantities in terms of the conserved; rather, one must perform a root-finding procedure to determine the values of $\boldsymbol{w}$ that give rise to the current state of the system, $\boldsymbol{q}$.

There are a number methods for this; see, e.g., Amano (2016); Dionysopoulou et al. (2013), and Palenzuela et al. (2009), of which we have implemented the latter two. In contrast to ideal RMHD, which requires a two-dimensional root-find to determine the primitive variables (Antón et al. 2010), resistive RMHD only requires a single residual to be minimized. In order to minimize the residuals given in Dionysopoulou et al. (2013) and Palenzuela et al. (2009), we use Newton–Raphson gradient descent. We find that in some instances one method may converge whereas the other may not, so implementing both residuals allows us to evolve a system with more confidence that the primitive variables will be found. In the vast majority of cases, however, either method will suffice.

Whichever primitive recovery method is implemented, the absolute tolerance to which the residual is to be minimized is important. For example, when evolving the two-fluid model, numerical errors in the velocity generated in the primitive recovery result in further errors in the charge current density. These errors are then multiplied by the square of the plasma frequency, ω_p^2, in the source term, which can be as large as 10^6 in the problems we present here. As a result, small errors accumulated during the conservative-to-primitive variable transformation are amplified in a manner that does not occur in single-fluid descriptions. It is for this reason that the tolerance is set to 10^{-13} for the two-fluid primitive recovery procedure in METHOD. This limit on the tolerance has implications for the possible accelerations that can be achieved; see Section 3.3.2.

In order to proceed as in the resistive, single-fluid RMHD case for the two-fluid model, we must first split the conserved fields into contributions from each species. For the relativistic mass-energy density, D, we see that

$$D_e = \frac{\bar{D} - \mu_p D}{\mu_e - \mu_p}, \quad \text{and} \quad D_p = \frac{\bar{D} - \mu_e D}{\mu_p - \mu_e}. \tag{7}$$

For the momentum and kinetic energy terms, we must first subtract the EM field contribution, as is normal in resistive, single-fluid RMHD. For example, we can define $\tilde{S}_i = \bar{S}_i - \epsilon_{ijk}E^jB^k$ and

Table 1
General Butcher Notation for Runge–Kutta Schemes

$\tilde{c}$	$\tilde{A}$	c	A
	$\tilde{w}^T$		w^T

then proceed as before with

$$\tilde{S}^i{}_e = \frac{\bar{S}^i - \mu_p \tilde{S}^i}{\mu_e - \mu_p}, \quad \text{and} \quad \tilde{S}^i{}_p = \frac{\bar{S}^i - \mu_e \tilde{S}^i}{\mu_p - \mu_e}. \tag{8}$$

Now, $\tilde{S}^i{}_s$ represents the purely hydrodynamical momentum of species s in the ith direction, i.e., $\tilde{S}^i{}_s = \rho_s h_s W_s^2 v^i{}_s$.

With these, we can continue the primitive recovery as in the resistive, single-fluid RMHD for each fluid species separately. We shall see that this algorithm demands a significant amount of computation, especially when incorporated with the IMEX schemes of the next section.

2.2. System Evolution

Here, we detail the numerical schemes that solve the equations of motion presented in the previous sections. METHOD uses a high-resolution shock-capturing approach, and provides a number of different options for how to evolve a simulation. In what follows, we will only describe those methods used in generating the results for this work.

2.2.1. IMEX Schemes

The equations of motion for both resistive models resemble a conservation equation with the addition of a (possibly) stiff source term—$\sigma \to \infty$ for the single-fluid description, and e.g., $\omega_p \to \infty$ for the two-fluid description. This is also the case for more general multi-fluid models that are of interest to neutron star simulations (Andersson et al. 2017a, 2017b, 2017c). In general, these kinds of systems will require the use of implicit time integrators to allow for the management of the large source terms. We implement the implicit–explicit, strong-stability-preserving, Runge–Kutta integrators of Pareschi & Russo (2004).

Redefining systems (1) and (5) as

$$\partial_t \boldsymbol{q} = \boldsymbol{\mathcal{L}}(\boldsymbol{q}) + \boldsymbol{\Psi}(\boldsymbol{q}), \tag{9}$$

where we have defined the numerical flux function, $\boldsymbol{\mathcal{L}}(\boldsymbol{q}) = -\partial_k \boldsymbol{f}^k(\boldsymbol{q})$, and the source term as $\boldsymbol{\Psi}(\boldsymbol{q})$, an IMEX scheme takes the general form

$$\boldsymbol{q}^{(i)} = \boldsymbol{q}^n + \Delta t \sum_{j=1}^{i-1} \tilde{a}_{ij} \boldsymbol{\mathcal{L}}(\boldsymbol{q}^{(i)}) + \Delta t \sum_{j=1}^{\nu} a_{ij} \boldsymbol{\Psi}(\boldsymbol{q}^{(i)}), \tag{10}$$

$$\boldsymbol{q}^{n+1} = \boldsymbol{q}^n + \Delta t \sum_{j=1}^{\nu} \tilde{w}_j \boldsymbol{\mathcal{L}}(\boldsymbol{q}^{(i)}) + \Delta t \sum_{j=1}^{\nu} w_j \boldsymbol{\Psi}(\boldsymbol{q}^{(i)}). \tag{11}$$

The matrices $\tilde{A}$ and A, with components $\tilde{a}_{ij}$ and a_{ij}, are constructed in such a way that the resulting equations for the flux contribution are explicit, and conversely are implicit for the source terms. Implicit and explicit forms for Equation (10) are enforced by ensuring zero and non-zero diagonal components in the respective matrices, $\tilde{A}$ and A. Therefore, $\tilde{a}_{ij} = 0$ for $i \geqslant j$, and $a_{ij} = 0$ for $i > j$.

The coefficients are derived by comparing the numerical solution with the Taylor expansion of the exact solution, as in other RK methods. Using the standard Butcher notation (Table 1)

Table 2
The Butcher Notation for Second-order SSP-RK Coefficients

0	0	0	γ	γ	0
1	1	0	$1-\gamma$	$1-2\gamma$	γ
	1/2	1/2		1/2	1/2

Table 3
The Butcher Notation for Third-order SSP-RK Coefficients

0	0	0	0	γ	γ	0	0
1	1	0	0	$1-\gamma$	$1-2\gamma$	γ	0
1/2	1/4	1/4	0	1/2	$1/2-\gamma$	0	γ
	1/6	1/6	2/3		1/6	1/6	2/3

the second-order accurate SSP2(222), and third-order accurate SSP3(332) coefficients are given in Tables 2 and 3, respectively, where $\gamma = 1 - 1/\sqrt{2}$, and the notation SSP $k(s, \sigma, p)$ refers to the order of the explicit part, k, number of implicit and explicit stages, s and σ, and order of the IMEX scheme, p.

The implicit stages given by Equation (10) require a root-finding procedure of the same dimensionality as the conserved vector $\boldsymbol{q}$. A guess is made for the solution $\boldsymbol{q}^{(i)}$, say $\boldsymbol{q}^{(*)}$, and the difference of the left and right sides of Equation (10) is computed as the residual to minimize. We minimize Equations (10) using the CMINPACK library, which we discuss in more depth in Section 3.3.2.

To determine the flux and source vectors for this guess at each iteration we require the values of the primitive quantities that correspond to the state $\boldsymbol{q}^{(*)}$. This process itself requires a root-finding procedure as mentioned in Section 2.1.3. As a result, the majority of the simulation time is spent performing the conservative-to-primitive variable transformation. This is important when optimizing the performance of the code.

2.2.2. Flux Reconstruction

There are numerous procedures for determining the numerical flux function, $\boldsymbol{\mathcal{L}}(\boldsymbol{q})$, with varying complexities and advantages/disadvantages. Common methods, such as the HLL-type schemes, are widely used but can become computationally expensive when moving to higher orders. Therefore, to reduce the cost of the reconstruction and to allow an easy extension to multiple dimensions and multiple physics models, we implement the method of FVS, proposed in Shu (1997).

In FVS, the flux of the conserved quantities through cells is considered to be composed of purely left- and right-going components. The components are then interpolated across a section of the domain to retrieve the reconstructed values of the left- and right-going pieces of the flux at the cell boundaries, and the differences taken on either side of a cell face to determine the net flux at that face.

To decompose the flux into purely left- and right-going pieces, we use the Lax–Friedrichs splitting,

$$\boldsymbol{f}_i^{\pm} = \frac{1}{2}(\boldsymbol{f}_i \pm \alpha \boldsymbol{q}_i), \tag{12}$$

where α is the maximum wave speed across the reconstruction ($\alpha = c = 1$, in MHD), and $\boldsymbol{f}_i$ is the flux vector at the center of cell i due to the state given by $\boldsymbol{q}_i$, with $\boldsymbol{f}_i^+ \geqslant 0$ and $\boldsymbol{f}_i^- \leqslant 0$.

The values of the flux at the cell faces are generated using a reconstruction that prevents (limits) numerical oscillations at shocks. There are a number of options for this, including targeted, essentially non-oscillatory reconstructions (Fu et al. 2017), but we find that the third-order WENO3 scheme of Shu (1997) works well.

3. Method

In this section we outline some of the design features of the relativistic and GPU-accelerated, multi-physics code METHOD.

METHOD is an object-oriented, C++ application for the evolution and testing of different MHD models. It is designed to allow maximum ease with which to test different features of plasma modeling using MHD, including flux reconstructions, time integrators, physics models, and initial conditions, and to assess the feasibility of GPU-implementations to ease computational demands. Extending METHOD to allow execution on graphics cards is achieved using NVIDIA's CUDA, version 9.0; see Section 3.2.

To ensure that tolerances of 10^{-13} are achieved for the primitive recovery, as discussed in Section 2.1.3, we have had to ensure that all data is stored in METHOD as a double-precision, floating-point number. This is in contrast to other GPU-capable MHD solvers such as Zink (2011), in which parallel speed-ups can be improved by relying only on single-precision accuracy. As a result, this will limit the performance improvements that are possible with the more complex, multi-fluid models, as double-precision is required to maintain stable evolutions.

3.1. Parallel Implementation

To investigate the possibility of performance improvements by utilizing graphics cards, the most computationally intensive tasks have been ported to the GPU. This form of design implementation is sub-optimal in the long term, as simulation data generally reside on the CPU and must be copied as needed to the GPU. This presents a limitation in the possible speed-up, as host-to-device and device-to-host memory transfers are notoriously slow (Nickolls et al. 2008; Bauer et al. 2011). It does, however, ease the process of adapting existing code to become GPU-capable. A more computationally efficient design would be to keep all data on the GPU throughout the simulation and only copy data to the host for outputting. Memory limitations can be handled, along with additional performance improvements, by utilizing multiple graphics cards. Data transfer between devices can then be done using CUDA-aware MPI calls. Future iterations of METHOD will be designed in such a way.

In order to achieve good parallel speed-ups by porting functions to the GPU, one needs to identify the bottlenecks in the simulation. For the two resistive models presented here, this is the root-finding procedure required by the IMEX schemes, and specifically the conservative-to-primitive variable transformation required in each iteration. We find that, depending upon the stiffness of the problem, an increase in execution time in excess of $10\times$ is not uncommon for resistive models using IMEX schemes when compared to ideal RMHD with explicit time integrators.

To maximize the chances of performance improvements, we therefore port the time integrator, and any functions it implements, to the GPU. This includes the implicit root-find for the IMEX integrator, flux vector calculations, source vector calculations, and, crucially, the primitive recovery.

3.2. CUDA Overview

The power of general purpose GPU (GPGPU) programming lies in the number of floating-point operations per second (FLOPS) permitted by modern GPUs (Nickolls et al. 2008). Graphics cards typically have a few thousand cores per device—where a core is a single processor, capable of executing one instruction at a time. The immense number of cores available on a GPU means that they lend themselves very well to scientific programming, in which there are often a large number of operations to be performed independently of each other.

Cores are batched together on the GPU (or device) into groups of 32, called streaming multiprocessors (SMs). Batches of 32 threads, called warps, are executed concurrently in which the same instruction is applied by the threads in the warp on different data. Numbers of warps are further abstracted into blocks, and a number of blocks are dispatched on the device at a time. The distinction of threads into blocks, and blocks into grids, eases thread recruitment and management of the data the threads are operating on. The optimum configuration of threads and blocks is discussed in Section 3.3.2.

While it is easy to run a program on a graphics card, running the program efficiently and achieving good parallel speed-ups can be difficult. Problems that are suited to GPU execution tend to require a large number of compute operations on a small data set, in addition to each operation being largely independent of each other. While there may be some interdependence between threads (e.g., during the flux reconstruction), fluid and MHD simulations are well suited to parallel programming architectures, on account of the large amount of work required to evolve a single cell in the domain.

3.3. Host and Device Memory

When programming for the CPU, developers rarely need to have an in-depth understanding of the underlying memory management. Modern day computers are extremely efficient at optimizing access to data in host memory (RAM), and it is often sufficient to allocate memory and allow the CPU to automatically use caching to speed-up performance. The same cannot be said when considering GPGPU programming, however.

NVIDIA graphics cards have multiple kinds of memory spaces that the programmer must be familiar with in order to achieve efficient execution. Good performance improvements are achieved only when the correct balance of the various memory resources is used, and getting this step wrong can significantly hamper possible improvements. We will now explain some of the strategies applied in METHOD to optimize memory management.

3.3.1. Memory Coalescence

When copying data between the CPU (or host) and device, one must ensure that the data are aligned in a suitable way. Transferring data to device memory essentially boils down to copying a single, one-dimensional array, and in order to optimize this process the developer must be familiar with how data is stored in memory and how the data is accessed for a computation.

In METHOD, data are aligned in the following way: the conserved vector is an array with dimensions $N_{\rm cons} \times N_x \times N_y \times N_z$, which we can access using `cons[var, i, j, k]` where `var, i, j, k` access the `var`th conserved variable in the `i`th, `j`th, and `k`th cell in the x, y, z directions, respectively. Due to the way in which memory is allocated in C++, all data are aligned first in the z-direction, followed by y, x, and then the variable, var—this means that all data in the z-direction are contiguous in memory.

As the flux reconstruction requires interpolating data between neighboring cells, this is easily performed in the z-direction, as data are already contiguous in memory. Reconstructing in the x- and y-directions, however, requires us to rearrange the original array such that data are contiguous in the x- and y-directions, respectively. This rearranging of data is also necessary when copying data to the device for the IMEX integrator, as in this case, data must be contiguous in the conserved variables, var.

The data are reorganized on the CPU so that they can be copied in a contiguous array to the device for operation. To reduce the performance hit this reorganizing has, we use OpenMP (OpenMP Architecture Review Board 2015) to parallelize this process. Another possible option to reduce this overhead is to copy all data to the device in their original order and allow threads to copy the relevant data from global memory to shared memory into the correct order. Initial efforts to achieve this slowed simulations by many factors, and instead we opted to stick with the CUDA/OpenMP hybrid method. In Section 4 we discuss the resulting performance impact.

3.3.2. Memory Optimization

There are three kinds of device memory that are utilized in METHOD—global, shared, and register—each with different sizes and latencies. Memory latency is the amount of time it takes from a thread requesting some data, to the data being available for computation. The different memory spaces have different latencies due to the distance from where the threads operate, the SM, to the location of the data on the device.

Global memory is the largest block of memory, and likewise the slowest to access. When data are copied to the device from the host, they are copied into global memory where every thread can access it. Shared memory is smaller in size, typically ≈96 KB per SM, but around 100× faster to access. Threads in the same block have access to the same shared memory, but cannot see data lying in another block's shared memory. This restriction means that when threads require data from their neighbors, one must overlap data between blocks. Finally, register memory is thread specific, and around 100× quicker than shared memory.

Handling memory management well in GPGPU programming is the difference between order of magnitude speed-ups or slow-downs, and often a large amount of trial and error testing is required until an optimal balance between shared and global memory is found. The same can be said in terms of the configuration of threads and blocks—a large number of threads in a block will limit the amount of shared memory per thread that is available, but a small number of threads may reduce the opportunity to hide memory latency behind useful computation. The optimal configuration will always be problem specific.

For the resistive RMHD models presented here, the best strategy we found is to maximize usage of shared memory for only 32 threads per block. This shared memory is specifically allocated for use in the primitive recovery, as these variables are accessed a very large number of times throughout the course of the IMEX scheme, and the benefits of having quicker

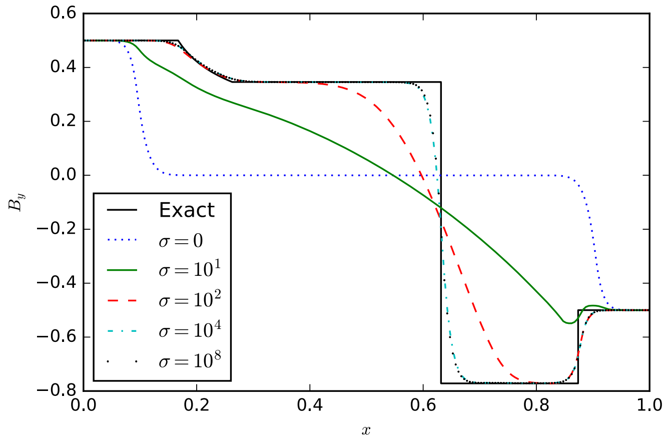


Figure 1. y-component of the magnetic field for the single-fluid, resistive RMHD model using the Brio–Wu initial data—as σ increases, the system tends toward the ideal RMHD solution. The system is evolved until $T = 0.4$ for $N_x = 200$ cells.

access to these variables outweighs the benefits of having them lie in global memory with the possibility of hiding memory latency within blocks. In order to hide latency between blocks, we therefore choose the maximum number of blocks possible with the available memory.

The most notable difference between implementing ideal and resistive MHD models on the GPU is the necessity of an implicit integrator to maintain stability in stiff regions. The process of implementing an implicit step requires some N-dimensional root-finding procedure, in which N is the size of the conserved vector. The package used for this process in METHOD is adapted from CMINPACK,[3] a C version of the widely used Fortran package, MINPACK.

CMINPACK requires the use of a work array of a size no less than $N(3N + 13)$ double-precision, floating-point numbers. The work array provides the CMINPACK routines with resources to determine the numerical Jacobian of the system, error norms, etc., and thus requires data that will be accessed multiple times throughout a single step. Ideally, this would lie in shared memory to minimize latency, but due to the size of both resistive systems and the limitations imposed by the size of the shared memory, the work array must be allocated in global memory. This issue is specific to systems that require implicit integrators, and does not occur for ideal RMHD models. As some of the data that are frequently required are located in global memory, one would expect greater latency when using implicit over explicit integrators, and thus for resistive over ideal MHD models.

4. Validation and Performance

In this section we begin by validating the solutions generated by METHOD for some well-established MHD and fluid test problems. Following this, we present the performance improvements achieved by execution on different generations of NVIDIA graphics cards, and show that parallel speed-ups of well over an order of magnitude are possible.

The range of initial data available in METHOD extends beyond what is presented here—we only show some results, but note that the solutions agree with the literature in every case for all of the models implemented.

[3] Original source: https://github.com/devernay/cminpack.

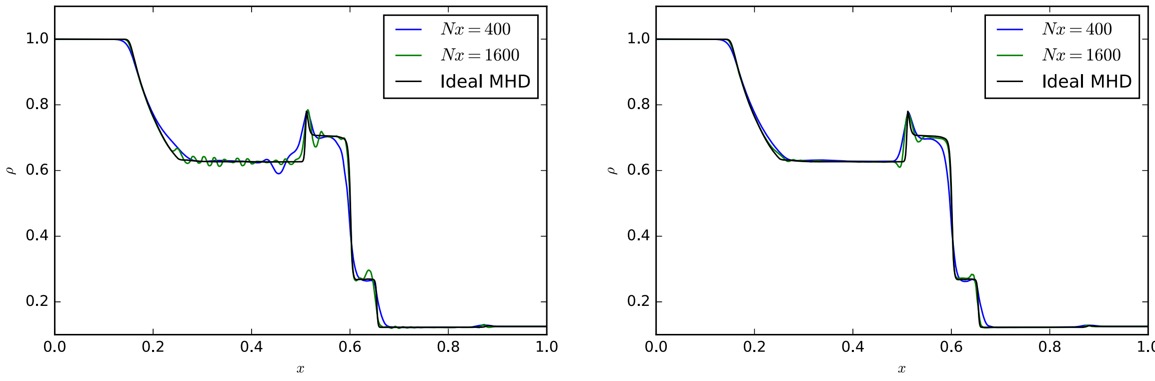


Figure 2. Rest-mass density for the two-fluid Brio–Wu shock tube test. Left: the results for the case of an electron–positron, pair plasma, i.e., $\mu_p = 10^3 = \mu_e$. Right: an electron–proton plasma of $\mu_p = 10^3 = \mu_e/100$. For both figures, the solution generated by the single-fluid, ideal MHD model is shown for reference. We have used up to 1600 grid points and evolved the system until $T = 0.4$. Due to the high resolution, we can see the two-fluid effect manifest as perturbations upon the single-fluid result—this effect is more pronounced for the higher-resolution run, as the effect is more easily resolved, and less pronounced for the non-pair plasma (right) due to the reduced skin depth (by a factor of 100×).

All tests presented are evolved with the SSP2(222) integrator unless stated otherwise.

4.1. Brio–Wu Shock Tube

A common setup that demonstrates shock-handling capabilities is the one-dimensional Brio–Wu shock tube test (Brio & Wu 1988). The domain is divided into two states, separated by a discontinuity, and at $t = 0$ the partition is removed. The two states then begin to interact and a series of waves propagates through the system. This problem reduces to a single Riemann problem, so there exists an exact solution with which we can compare our solutions. The exact solution in the ideal RMHD limit, $\sigma \to \infty$, is generated using the code of Giacomazzo & Rezzolla (2006).

4.1.1. Single-fluid

To verify the behavior in the single-fluid, resistive regime we use the setup from Palenzuela et al. (2009), and vary the magnitude of the conductivity. The initial state is given by Equation (13) and we use $\Gamma = 2$ for consistency with Palenzuela et al. (2009) and Dionysopoulou et al. (2013):

$$(\rho,\, p,\, B_y) = \begin{cases} (1,\, 1,\, 0.5) & \text{for } 0 \leqslant x < L/2, \\ (0.125,\, 0.1,\, -0.5) & \text{for } L/2 \leqslant x \leqslant L. \end{cases} \tag{13}$$

4.1.2. Two-fluid

For the two-fluid RMHD model we use the initial data from Amano (2016), given by Equation (14). At high enough resolutions, we can see the onset of the two-fluid effect manifest as oscillations in the fluid's density:

$$(\rho,\, p,\, B_x,\, B_y) = \begin{cases} (1,\, 1,\, 0.5,\, 1) & \text{for } 0 \leqslant x < L/2, \\ (0.125,\, 0.1,\, 0.5,\, -1) & \text{for } L/2 \leqslant x \leqslant L. \end{cases} \tag{14}$$

The results for both resistive RMHD models are in excellent agreement with the literature (see Figures 1 and 2), and show that METHOD can avoid numerical oscillations and instabilities for discontinuous data.

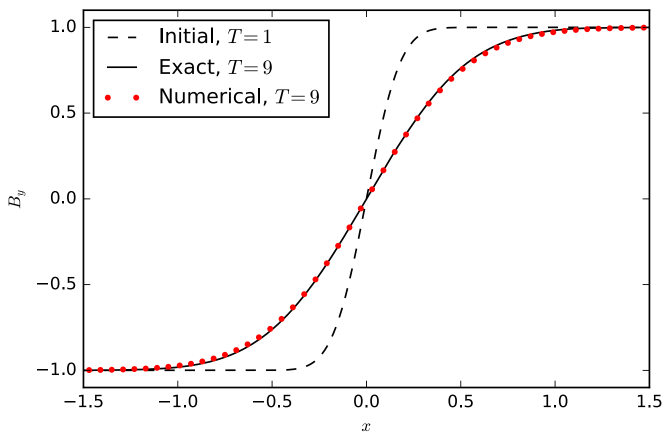


Figure 3. Comparison of the exact and numerical solutions for the self-similar current sheet for a two-fluid model using $N = 50$ cells and simulating a pair plasma, $\mu_p = 10^3 = -\mu_e$.

4.2. Self-similar Current Sheet

A useful test of the resistive nature of MHD models was given first in Komissarov (2007). The system is set in hydrodynamic equilibrium with $(p,\, \boldsymbol{v},\, \rho) = (50.0,\, 0,\, 1.0)$. If a magnetic field of $(B_x,\, B_y,\, B_z) = (0,\, B(x,\, t),\, 0)$ is applied, the resulting dynamics of the field are given by the diffusion equation,

$$\partial_t B - \eta \partial_x^2 B = 0, \tag{15}$$

a solution of which is given by the self-similar error function:

$$B_y(x,\, t) = B_0 \operatorname{erf}\left(\frac{\sqrt{\sigma x^2}}{2t}\right). \tag{16}$$

At $t = 1.0$, we initialize the electromagnetic fields with $(\boldsymbol{E},\, B_0) = (0,\, 1.0)$ so that the magnetic pressure is much less than the hydrodynamic pressure, and use a moderate conductivity $\sigma = 100$. The B_y solution at $T = 9.0$ is shown in Figure 3 for the two-fluid model for $N = 50$ cells—even at such low resolutions the exact and numerical solutions are virtually indistinguishable.

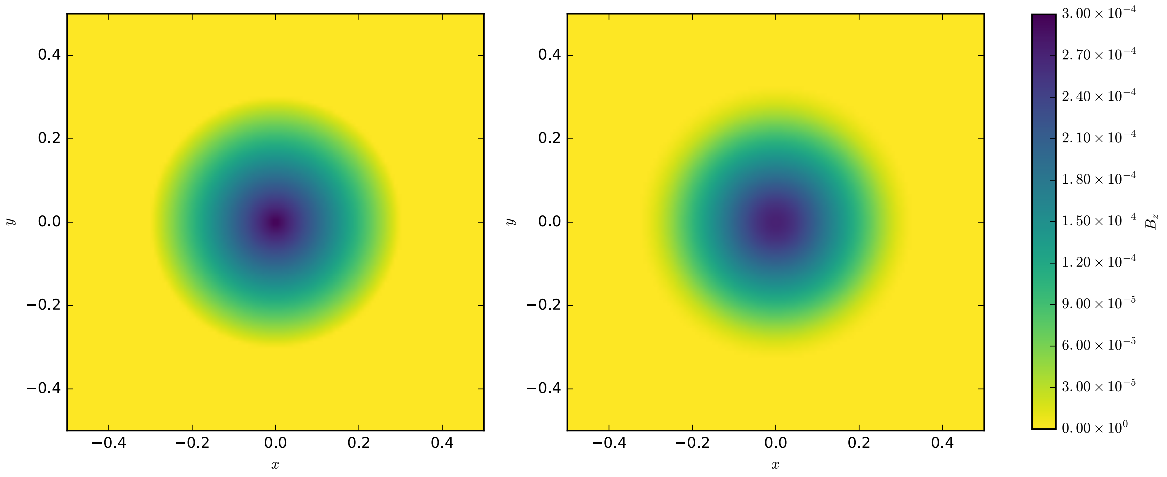


Figure 4. Initial state (left) and final state after one crossing time for the z-direction magnetic field, B_z, using 128 × 128 grid points for the field loop advection test.

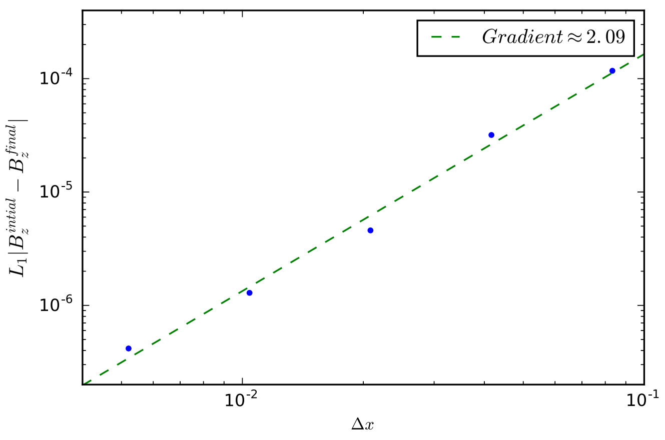


Figure 5. Convergence for a range of resolutions for the field loop advection test. The convergence is as expected at second order.

4.3. Field Loop Advection

Convergence tests are useful tools when assessing code behavior. It is expected that as the resolution of the simulation is increased, the numerical solution should converge to the true solution as a function of the resolved scale. To show the numerical convergence of METHOD, we present the field loop advection test, adapted from Gardiner & Stone (2008). In this setup, the fluid is set in hydrodynamic equilibrium and a weak magnetic field loop are advected through the system. Comparing the solution after one crossing time to the initial data for a range of spatial resolutions gives an indicator of the order of convergence of the algorithm.

The hydrodynamic system is initialized with $(\rho, v_x, v_y, v_z, p) = (3, 0.1, 0.1, 0, 1)$ in a two-dimensional domain with boundaries at $(-0.5, 0.5) \times (-0.5, 0.5)$, and composed of $N \times N$ grid points. The z-direction magnetic field is then set as

$$B_z = \begin{cases} B_0(R - r) & \text{for } r \leqslant R \\ 0 & \text{for } r > R \end{cases}, \qquad (17)$$

with $B_0 = 10^{-3}$.

For this test, we compared initial and final states after one crossing time, $T = 10$, with periodic boundaries. The results are shown in Figure 4 for the single-fluid resistive model in the stiff limit, $\sigma = 10^5$, and with $\Gamma = 2.0$. The error is calculated with the L_1-norm for all cells in the system, and plotted as a function of the spatial resolution, where we note that $\Delta x = \Delta y$. Figure 5 shows the expected second-order convergence, which is limited by the order of the time integrator.

4.4. Kelvin–Helmholtz Instability (KHI)

A common instability that occurs in real-life applications of numerical fluid codes, and believed to be instrumental in forming strong magnetic fields in mergers (Zhang et al. 2009; Obergaulinger et al. 2010; Kiuchi et al. 2015a), is one of differentially flowing fluids, the so-called KHI. This two-dimensional problem concerns the growth of perturbations along a boundary layer between two fluids with relative velocities, which evolve to the recognizable vortices seen in some cloud formations.

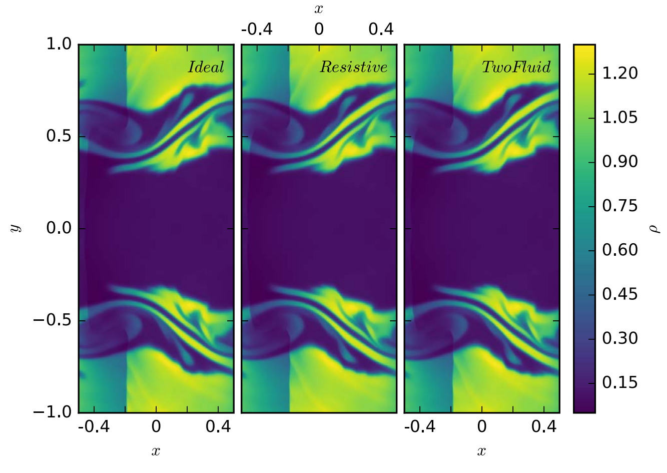
The initial data have been adapted from Mignone et al. (2009). We initialize a two-dimensional domain, where $x, y \in [-0.5, 0.5], [-1.0, 1.0]$, with the following properties:

$$v_x = \begin{cases} v_{\text{shear}} \tanh\left(\dfrac{y - 0.5}{a}\right) & \text{if } y > 0.0 \\ -v_{\text{shear}} \tanh\left(\dfrac{y + 0.5}{a}\right) & \text{if } y \leqslant 0.0, \end{cases} \qquad (18)$$

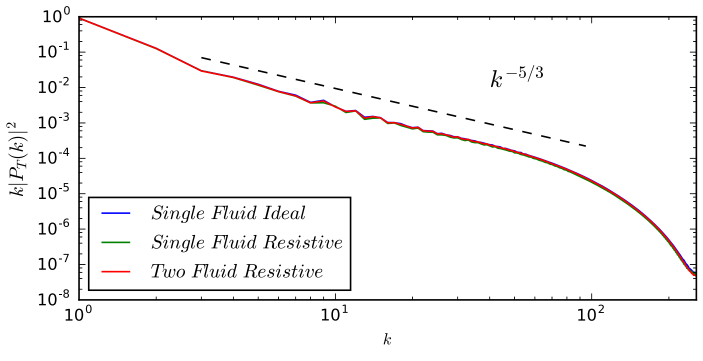
$$v_y = \begin{cases} A_0 v_{\text{shear}} \sin(2\pi x) e^{\frac{-(y-0.5)^2}{l^2}} & \text{if } y > 0.0 \\ -A_0 v_{\text{shear}} \sin(2\pi x) e^{\frac{-(y+0.5)^2}{l^2}} & \text{if } y \leqslant 0.0, \end{cases} \qquad (19)$$

$$\rho = \begin{cases} \rho_0 + \rho_1 \tanh\left(\dfrac{y - 0.5}{a}\right) & \text{if } y > 0.0 \\ \rho_0 - \rho_1 \tanh\left(\dfrac{y + 0.5}{a}\right) & \text{if } y \leqslant 0.0, \end{cases} \qquad (20)$$

in which the shear velocity is $v_{\text{shear}} = 0.5$, with a thickness of $a = 0.01$, $(\rho_0, \rho_1) = (0.55, 0.45)$, and the amplitude perturbation in y-direction is $A = 0.1$ over a length scale of $l = 0.1$. The system is initially at a constant pressure of $p = 1.0$ with $\Gamma = 4/3$ and a magnetic field perpendicular to the fluid flow of $B_z = 0.1$, the conductivity is set at $\sigma = 10$, and a charge-mass ratio of $\mu_p = -\mu_e = 10^2$ is used for the two-fluid model.



(a)



(b)

Figure 6. Results for the Kelvin–Helmholtz instability simulation. Figure (a) shows the final state of the density for the ideal and resistive single-fluid and two-fluid models at $T = 6.0$, and Figure (b) the kinetic energy power spectrum for each model at $T = 3.0$. All models follow Kolmogorov's 5/3 power law. The hydrodynamic behavior for each model is virtually indistinguishable due to the low magnitude of the magnetic fields and the relatively coarse resolution. All models are evolved with 512×1024 cells.

Figure 6 shows the density distribution at $T = 6.0$ for each of the models (left). The magnetic field strength is low and thus does not greatly impact the behavior of the fluid, resulting in near identical final states for all models. We also show the kinetic energy power spectrum, Figure 6(b), following the procedure in Beckwith & Stone (2011) and note that it follows Kolmogorov's 5/3 power law, as expected.

4.5. Performance

Our attention now turns to the performance of the CPU and GPU - implementations of METHOD, and the potential benefits of GPU-capable, resistive MHD codes. For the comparison, we make use of the IRIDIS 5 HPC cluster at the University of Southampton—the GPU nodes possess a number of NVIDIA GTX1080 (Pascal architecture) (NVIDIA GeForce GTX1080) and V100 (Volta architecture) (NVIDIA Tesla V100 GPU Architecture 2017) graphics cards. We perform our analysis on a single GTX1080, and a single V100 card to assess performance differences between the two architectures, and compare these against a benchmark, CPU implementation on an INTEL XEON 6138 GOLD using a single core.

To compare the execution times of the serial and parallel implementations, we measure the wall clock time of the main evolution loop for 10 iterations. This execution does not involve any time spent in setting up the domain or initial data, or writing data to the disk, and thus is representative of the differences one would expect in the use of these methods for large-scale merger simulations where the majority of the execution time is spent on the evolution of the system (as opposed to initialization or output of data).

We use two different initial data for the comparison—the Brio–Wu shock tube test and the more complex KHI instability, both described in the previous section. Furthermore, as we are only concerned with the GPU - implementations of the two resistive models, we use the IMEX schemes mentioned in Section 2.2.1—results from both the SSP2(222) and SSP3 (322) integrators (Pareschi & Russo 2004) are presented. All simulations are run using double-precision floating-point accuracy as per the discussion in Section 2.1.3.

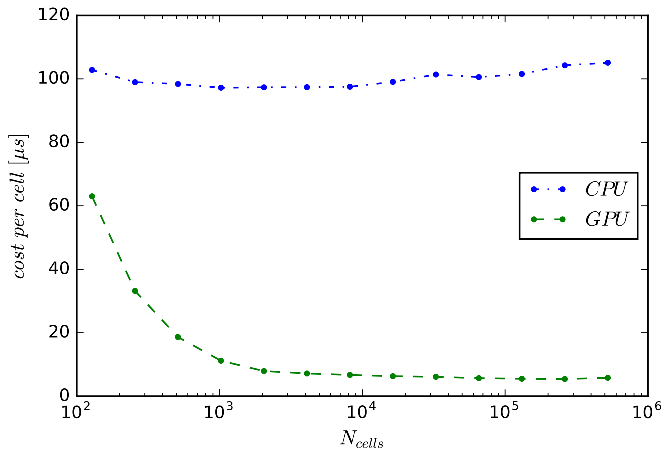


Figure 7. Measure of the cost-per-cell (execution time in μs) on a V100 graphics card and the CPU for the Brio–Wu test using the SSP3 integrator. Execution on the CPU is independent of the domain size, while execution time on the device decreases until the GPU is saturated.

The serial execution time does not vary much between runs as differences depend only upon the current state of the CPU. The execution time for the parallel implementation, however, depends on a number of factors, such as usage of shared memory and registers, number of host-device memory copies, and the configuration of threads and blocks. As such, presented here are the optimal configurations that minimize the execution time on the graphics card. We find that the optimal configurations is to minimize the number of threads in a block to 32, as discussed in Section 3.3.2, to allow sufficient usage of shared memory for the primitive recovery. We then maximize the number of blocks per grid to hide latency, relax the condition that serial and parallel output must exactly match (while keeping physically valid results), compile device code with the `−O3` flag and switch off CUDA's fused-multiply-add functionality.

Figure 7 shows the average cost-per-cell for the Brio–Wu simulation when using the SSP3 integrator on the CPU and a V100. As expected, the cost-per-cell for the CPU implementation

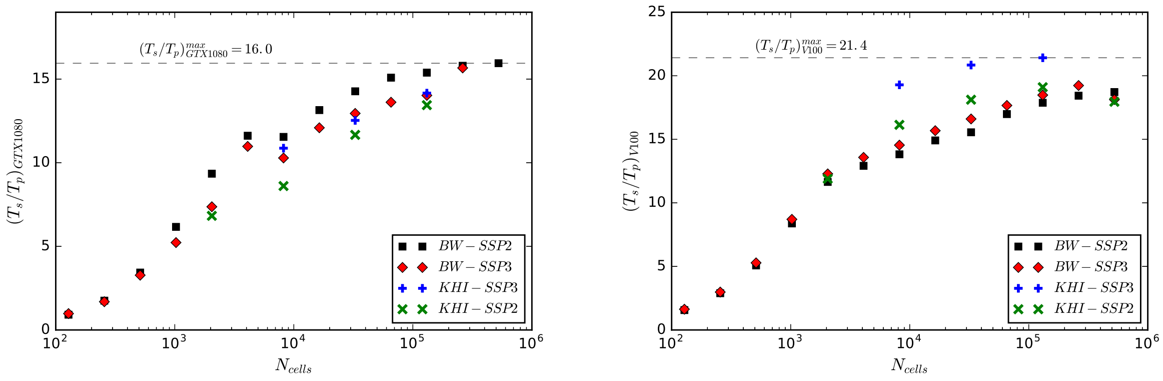


Figure 8. Measure of parallel speed-up, $T_{\text{serial}}/T_{\text{parallel}}$, for the GTX1080 (left) and V100 (right) graphics cards. The squares and diamonds correspond to the SRRMHD single-fluid Brio–Wu, while the pluses and crosses correspond to the two-fluid Kelvin–Helmholtz instability, respectively. The speed-up depends strongly upon the total number of cells, reaching a maximum of $21\times$ before memory resources on the device are exhausted. The maximum parallel speed-up for each card is indicated by the dashed line.

is independent of the size of the domain, and all data lies in the same memory (RAM), so access and computation of data are common among all cells. Instead, looking at the GPU implementation, we can see that as the number of cells in the simulation increases, the cost-per-cell reduces drastically.

Smaller domains contain fewer cells, and thus fewer opportunities to hide memory latency behind useful computation. As a result, for small domains the problem is memory-bound—a significant amount of time is spent waiting for access to various locations in memory and as a result the threads lay idle. As we increase the size of the domain, the bottleneck then becomes the amount of computation required for each time step, and the problem is then compute-bound. We can see the transition of the problem from memory- to compute-bound in Figure 7 as the cost-per-cell begins to plateau for the largest systems. At this point, the GPU is saturated, and improvements may only be made via smarter memory management, or additional GPUs.

To see how this impacts the absolute speed-up due to the GPU implementation, we plot the ratio of the serial execution time and parallel execution time (parallel speed-up) for 10 time-steps as a function of the total number of grid points in the simulation, Figure 8. The results for two generations of cards, GTX1080 and V100, are shown. It is clear by comparing the two cards that the newer generation graphics card, the V100 with the Volta architecture, provides improved speed-ups. It can also be seen that there is little difference between the results of the initial data and the two IMEX schemes for the older (but still widely used) Pascal architecture. On the other hand, the new Volta architecture offers greater memory bandwidth (nearly $3\times$), and therefore a reduced memory latency—the result is to allow execution time to be dominated by computation. We see this as a much improved parallel speed-up, over $21\times$, specifically for the compute-intensive algorithms of the two-fluid RMHD models with the SSP3-IMEX scheme.

The previous result suggests that higher-order methods that require additional computation will achieve greater parallel speed-ups than lower-order alternatives. The additional computation required by an IMEX-SSP3(433) scheme, for example, which requires a total of four implicit stage and three flux reconstructions, would result in a greater proportion of the execution time being spent on the graphics card. As a result, the benefits of a GPU-implementation for simulations in which the higher-order schemes are used would be more apparent—merger simulations typically employ fourth order methods, for example.

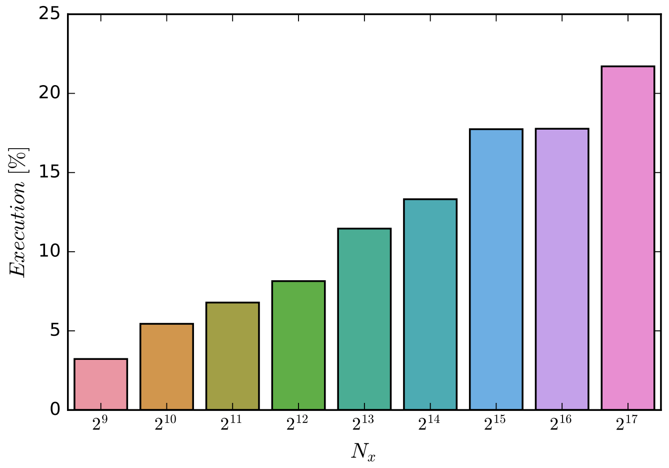


Figure 9. Cost of rearranging data for the IMEX scheme as a percentage of the total execution time. There is a strong dependence on the number of cells in the domain.

To gain insight into where the performance bottlenecks are in METHOD, we present a table with various profiling data, Table 4. Data were generated using the Brio–Wu initial data and a relatively large domain on $N_x = 131072$ grid points, the SSP3 integrator using third-order FVS reconstruction, and a conductivity of $\sigma = 10^4$. Even with execution on the device, the IMEX scheme makes up the largest proportion of the execution time. The next largest contributor is the rearrangement of data into contiguous arrays for the IMEX scheme, as discussed in Section 3.3.1. The fraction of time taken to rearrange the data is heavily dependent upon the domain size. The expected scaling of the rearrangement on the CPU follows the number of cells, N_x, but we have seen that the cost-per-cell of the main loop reduces with the number of cells. As a result, as the domain increases in size, more time is spent proportionally rearranging the data when compared to the total execution time (which is dominated by the execution on the device). We can see this visually in Figure 9 as a growing fraction of the execution time being spent reordering data. Large-scale simulations of neutron star mergers may evolve 10 times as many grid points as we show here, so a significant

Table 4
Performance Profile for the Brio–Wu Initial Data Using the SSP3 Integrator and $N_x = 2^{17}$ Cells

	Performance Profile				
Functionality	IMEX Rearrange	IMEX Root-find	FVS Rearrange	FVS Reconstruction	Boundary Conditions
Proportion of Execution (%)	21.7	55.3	1.0	2.2	0.2

Note. Results represent the percentage of the total execution time of the main loop. The rest of the execution time is composed of the correction stage, Equation (11), primitive recovery that is not within the implicit stages of the IMEX scheme, and data and memory management.

fraction of the execution time would be performing this rearrangement—in this case, a more efficient scheme for performing this would be highly beneficial.

The time spent rearranging data for the flux reconstruction, however, is significantly less than that for the IMEX scheme. As only the conserved and flux vectors need rearranging, this results in a total of three arrays being reordered in the FVS method. This is compared to a total of 28 arrays required for the SSP3 scheme, including the conserved, primitive, source, and flux vectors for each stage. As a result, there are fewer benefits to be had from optimizing the flux reconstruction.

5. Discussion

This work describes the development of a new, publicly available, GPU-accelerated, numerical, multi-physics RMHD code, METHOD. We have demonstrated the validity of its solutions, and investigated the available performance improvements made possible by porting compute-intensive functions to the GPU.

When evolving resistive single or multi-fluid RMHD models, one must use a time integrator that solves, at the least, the source terms implicitly—as a result, the majority of the execution time for these kinds of simulations is spent in these schemes. We have shown that the most commonly used class of integrators for this procedure, IMEX-RK schemes, can be accelerated by at least a factor of 21× by execution on the device. Furthermore, we argue that there are potentially greater improvements to be made for higher-order methods, as proportionally more time will be spent in the accelerated regions of the simulation.

The greatest improvements of performance are possible when managing the resources of shared memory correctly. The recovery of the primitive variables, necessary for each iteration of the implicit stage in the IMEX scheme, itself requires a root-finding procedure. To minimize memory latency, the arrays required for the recovery are kept in shared memory—this severely limits the number of threads per block that we can recruit but saves multiple factors in terms of execution time.

The available speed-ups of METHOD are limited by its design—METHOD has been extended to execute the most computationally expensive functions (time integration) on the device, and inherent in this is a number of host-to-device and device-to-host memory transfers. In order to copy data from contiguous arrays we are required to perform a rearrangement of the data on the host which takes up a significant fraction of the overall execution. To reduce this overhead, and to improve the potential parallel speed-ups, we note that a more efficient design would be to develop the GPU-capabilities from the outset. In this way, all data will be in device memory and only be transferred to the host for output. Organizing the simulation data in this way may allow threads to load data into contiguous arrays in parallel on the device, which for large domains may reduce the cost of the rearrangement significantly.

Furthermore, even with the acceleration of the time integrator by execution the device, the majority of time is still spent in the IMEX root-find. This means that optimizations in this region will likely produce further improvements. This could include reducing the size of the work array required by the root-finding procedure such that it can fit in shared memory, although greater improvements seem likely by keeping the primitive recovery variables in shared memory at the expense of the IMEX work array.

The size of the conserved vector has little effect on the possible speed-up. The single-fluid RMHD model, which has the smallest conserved vector presented, requires a work array for the time integration that is too large to fit in shared memory. As a result, the speed-up comes primarily from the primitive recovery variables lying in shared memory. The same is true for models with larger conserved vectors, so more complex, four-fluid models should have comparable accelerations—the required memory for the primitive recovery grows linearly with the conserved vector size, in contrast to the quadratic dependence for the time integrator. As a result, the methods employed here should transfer over to the more computationally demanding, general-relativistic multi-fluid models proposed in Andersson et al. (2017a).

In the future, we aim to combine the results of what has been presented here with adaptive mesh refinement (AMR) techniques to utilize multiple GPUs in an MPI-OpenMP-CUDA hybrid code. Resistive multi-fluid models present us with a huge step in computation required to evolve a system. The increased size of the conserved vector and the necessity of implicit–explicit integrators for stable evolution mean that current implementations are not suitable for large-scale, merger simulations. With a hybrid code of this type, one can utilize the immense amount of compute power provided by GPU clusters, allowing the evolution of more physically accurate systems than has previously been possible.

The first release of METHOD used for generating these results is available through Zenodo (Wright 2018), and the latest iteration is publicly available on GitHub.[4]

A.J.W. thanks Anthony Morse, Philip Blakely and Timothy Lanfear for insightful discussions about CUDA implementation and optimization strategies. The authors acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work. We also gratefully acknowledge financial support from the EPSRC Centre for Doctoral Training in Next Generation Computational Modelling grant EP/L015382/1.

[4] https://github.com/AlexJamesWright/METHOD

Software: METHOD (Wright 2018), CMINPACK (https://github.com/devernay/cminpack), Matplotlib (Hunter 2007), seaborn (Waskom et al. 2014).

Appendix Two-fluid model

The flux vector and source term for the two-fluid, resistive RMHD model are as follows:

$$
\boldsymbol{f}^j(\boldsymbol{q}) = \begin{pmatrix} \rho_s W_s v_s^j \\ S_s^{ij} \\ S^j - D_s v_s^j \\ \overline{D}_s v_s^j \\ \overline{S}_s^j v_s^i + \delta^{ji} p_s \mu_s \\ (\overline{\tau}_s + \mu_s p_s) v_s^j \\ \epsilon^{ijk} E_k + \delta^{ij}\phi \\ -\epsilon^{ijk} B_k + \delta^{ij}\psi \\ E^j \\ B^j \end{pmatrix}, \tag{21}
$$

$$
\boldsymbol{s}(\boldsymbol{q}) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \omega_p^2 [W E^i + \epsilon^{ijk} u_j B_k - \eta (J^i - \varrho u^i)] \\ \omega_p^2 [u_k E^k - \eta(\varrho - \varrho_0 W)] \\ 0 \\ -J^i \\ \varrho - \kappa\psi \\ -\kappa\phi \end{pmatrix}, \tag{22}
$$

with the relations

$$
\begin{pmatrix} S_s^{ij} \\ \varrho \\ J^i \\ W \\ u^i \\ \varrho_0 \\ \omega_p^2 \end{pmatrix} = \begin{pmatrix} \rho_s h_s W_s^2 v_s^i v_s{}^j + \delta^{ij} p_s - [E^i E^j + B^i B^j] + \delta^{ij}[E^2 + B^2]/2 \\ \mu_s \rho_s W_s \\ \mu_s \rho_s W_s v_s^i \\ \mu_s^2 \rho_s W_s / \omega_p^2 \\ \mu_s^2 \rho_s W_s v_s^i / \omega_p^2 \\ W\varrho - J_k u^k \\ \mu_s^2 \rho_s \end{pmatrix}. \tag{23}
$$

ORCID iDs

A. J. Wright https://orcid.org/0000-0002-5953-4221
I. Hawke https://orcid.org/0000-0003-4805-0309

References

Amano, T. 2016, ApJ, 831, 100
Andersson, N., Comer, G. L., & Hawke, I. 2017a, CQGra, 34, 125001
Andersson, N., Dionysopoulou, K., Hawke, I., & Comer, G. L. 2017b, CQGra, 34, 125002
Andersson, N., Hawke, I., Dionysopoulou, K., & Comer, G. L. 2017c, CQGra, 34, 125003
Antón, L., Miralles, J. A., Martí, J. M., et al. 2010, ApJS, 188, 1
Balsara, D. S., Amano, T., Garain, S., & Kim, J. 2016, JCoPh, 318, 169
Bauer, M., Cook, H., & Khailany, B. 2011, in SC 11 Proc. 2011 Int. Conf. for High Performance Computing, Networking, Storage and Analysis
Beckwith, K., & Stone, J. M. 2011, ApJS, 193, 6
Brio, M., & Wu, C. 1988, JoCP, 75, 400
Dedner, A., Kemm, F., Kröner, D., et al 2002, JoCP, 175, 645
Dionysopoulou, K., Alic, D., Palenzuela, C., Rezzolla, L., & Giacomazzo, B. 2013, PhRvD, 88, 044020
Dionysopoulou, K., Alic, D., & Rezzolla, L. 2015, PhRvD, 92, 084064
Font, J. A. 2008, LRR, 11, 7
Fu, L., Hu, X. Y., & Adams, N. A. 2017, JCoPh, 349, 97
Gardiner, T. A., & Stone, J. M. 2008, JCoPh, 227, 4123
Gawehn, E., Hiss, J. A., Brown, J. B., et al. 2018, Expert Opinion on Drug Discovery, 13, 579
Giacomazzo, B., & Rezzolla, L. 2006, JFM, 562, 223
Hunter, J. D. 2007, CSE, 9, 90
Kiuchi, K., Cerdá-Durán, P., Kyutoku, K., Sekiguchi, Y., & Shibata, M. 2015a, PhRvD, 92, 064059
Kiuchi, K., Sekiguchi, Y., Kyutoku, K., et al. 2015b, PhRvD, 92, 064034
Komissarov, S. S. 2007, MNRAS, 382, 995
Köppel, S. 2017, JPhCo, 1031, 012017
Lee, D., Dinov, I., Dong, B., et al. 2012, Computer Methods and Programs in Biomedicine, 106, 175
Mignone, A., Ugliano, M., & Bodo, G. 2009, MNRAS, 393, 1141
Nickolls, J., Buck, I., Garland, M., & Skadron, K. 2008, Scalable Parallel Programming with CUDA (New York: ACM), https://queue.acm.org/detail.cfm?id=1365500
Nouri, F. H., Duez, M. D., Foucart, F., et al. 2018, PhRvD, 97, 83014
NVIDIA GeForce GTX 1080 2016, Whitepaper
NVIDIA Tesla V100 GPU Architecture 2017, Whitepaper
Obergaulinger, M., Aloy, M. A., & Müller, E. 2010, A&A, 515, A30
OpenMP Architecture Review Board 2015, 359
Palenzuela, C., Lehner, L., Reula, O., & Rezzolla, L. 2009, MNRAS, 394, 1727
Pareschi, L., & Russo, G. 2004, JSCom, 25, 129
Price, D. J., & Rosswog, S. 2006, Sci, 312, 719
Radice, D. 2017, ApJL, 838, L2
Ruiz, M., Shapiro, S. L., & Tsokaros, A. 2018, PhRvD, 97, 21501
Shu, C.-w. 1997, in Advanced Numerical Approximation of Nonlinear Hyperbolic Equations (Springer: Berlin), 325
Waskom, M., et al. 2014, Seaborn, Zenodo, doi:10.5281/zenodo.12710
Wong, H. C., Wong, U. H., Feng, X., & Tang, Z. 2011, CPCom, 182, 2132
Wright, A. 2018, METHOD: Initial Public Release, Zenodo, doi:10.5281/zenodo.1404697
Zhang, W., MacFadyen, A., & Wang, P. 2009, ApJ, 692, 40
Zink, B. 2011, arXiv:1102.5202