Parallel Explicit/Implicit Time Stepping Scheme on Block-Adaptive Grids

Gábor Tóth, Darren L. De Zeeuw, Ying-Juan Ma, Tamas I. Gombosi, Kenneth G. Powell

Center for Space Environment Modeling, University of Michigan, Ann Arbor, Michigan, 48109-2143. (gtoth@umich.edu)

Abstract. This proceedings paper is mostly based on a much longer research paper Tóth et. al. (2006). We present a parallel explicit/implicit time integration scheme well suited for block-adaptive grids. Load balancing and the optimal choice of the time step for speed and robustness are discussed. The parallel efficiency of the scheme is demonstrated for a three-dimensional Hall magnetohydrodynamics problem.

1. Introduction

Numerical modeling of disparate spatial and temporal scales presents a formidable challenge to computational physics. Here we focus on space physics applications, in particular the Sun-Earth system, but there are similar challenges in many other areas. A physics based model of the Sun-Earth system consists of many different physics domains: the solar corona, heliosphere, magnetosphere, inner magnetosphere, radiation belts, plasma sphere, ionosphere, thermosphere, etc. The domains differ in spatial and temporal scales as well as in the physics. Our group at the University of Michigan has been working on building the Space Weather Modeling Framework (SWMF, Tóth et. al. 2005) to model this complex system accurately and efficiently. Efficiency is especially crucial for predicting space weather, i.e. the effects of solar eruptions on the Earth and its environment: the model has to execute significantly faster than real time to be able to provide useful predictions.

One can easily estimate the required grid resolution in the various regions of the Sun-Earth system. Here we only consider the physics domains that can be reasonably described with the equations of magnetohydrodynamics (MHD). One needs at least $1/100 R_S$ (solar radii, $R_S \approx 700,000 \text{ km}$) resolution in the solar corona model near the active region where the eruption is coming from. To model the magnetosphere near the active region it is necessary to resolve scales of $1/8 R_E$ (Earth radii, $R_E \approx 6,380 \text{ km}$) at least. On the other hand the distance between the Sun and the Earth is one astronomical unit (1 AU $\approx 150,000,000 \text{ km}$) that is about 20,000 and 200,000 times larger than the grid resolution needed near the active region and near the Earth, respectively. It is quite clear that one cannot use a uniform grid to model this 3 dimensional region, because the number of grid cells would be about $10^{15}$ or more. Therefore one needs to use a non-uniform and/or adaptive grid. The MHD components of the SWMF use block-adaptive grids.
The time scales are also quite challenging. It takes about 1 to 3 days for a coronal mass ejection to reach the Earth. On the other hand the time step of an explicit time integration scheme is limited to about 0.03 seconds near the Earth, since the fastest MHD wave speed in that region is about 30,000 km/s while the cell size is about 1000 km or smaller. To overcome this difficulty we have developed an efficient parallel implicit time stepping scheme that is the subject of this paper.

In general explicit time stepping becomes very inefficient when the time step is limited by numerical stability to orders of magnitude smaller than the time step required by the dynamics of the system. Such stiff problems occur for hyperbolic as well as parabolic equations. The stiffness can be due to very high wave speeds and/or very different cell sizes. For example in Hall MHD the speed of the whistler wave is inversely proportional to the wave length. The smallest wave length supported by the grid is proportional to the cell size, therefore the explicit time step will be proportional to the square of the cell size, which is even worse than the case for ordinary MHD. Another example is the use of cylindrical or spherical grids. Near the symmetry axis the grid cells become very elongated and their width is proportional to the radial cell size divided by the number of cells in the azimuthal direction ($r \Delta \phi \approx 2\pi \Delta r / N\phi$). Again the explicit time step is severely limited as the resolution is increased.

If the ratio of the time steps required by accuracy and stability is large (say greater than one hundred) switching to an implicit time integration scheme is likely to improve the efficiency significantly. Designing an efficient parallel implicit scheme for a block-adaptive grid is not a simple problem. There are many choices to be made that can affect the performance of the code. We will discuss these below.

2. Adaptive Mesh Refinement

Adaptive mesh refinement (AMR) is becoming a standard tool for resolving many orders of spatial scales. There are several variants of AMR grids: cell-based mesh refinement (e.g. Dinge & Woodward 2003), hierarchical patches (e.g. Berger & Colella 1989) and block-based mesh refinement (e.g. Powell et. al. 1999).

The block-adaptive grid approach is very similar to the cell-based AMR, except that the smallest unit that can be refined is a block with a fixed number of cells (for example $4 \times 4 \times 4$ cells in 3D). When a 3D block is refined, it is split into 8 octants (see Figure 1). Each octant forms a block with the same number of cells as the original block, but the resolution is increased by a factor of two. The resulting grid structure is an octree of blocks, and the equations are solved at the finest level only, i.e. on the leaves of the tree. The time step is either proportional to the grid resolution, or it can be the same for all the blocks. The blocks are surrounded with ghost cells to simplify inter-block communication. Any scheme developed for a uniform grid can be used, although algorithms requiring a very wide stencil will need many ghost cell layers, which can be expensive. Since the blocks have the same number of cells, load balancing the parallel execution is quite simple, especially if the time step is uniform. Note that the number of blocks is typically much larger than the number of processors. The typical
number of blocks is many thousands with sizes ranging from $4 \times 4 \times 4$ to $8 \times 8 \times 8$ cells depending on the application.

In this paper we concentrate on time integration algorithms designed for block AMR grids, although the schemes should carry over to AMR grids using hierarchical patches without any difficulty. Explicit time integration on a block AMR grid is relatively simple, especially when all the grid cells are advanced with the same time step. In many applications the number of cells (or blocks) at the finest spatial resolution dominates, thus using the same time step in all the cells is only moderately less efficient than advancing the blocks of different grid resolution with proportional time steps. Using uniform time steps reduces the complexity of the code and allows very simple load balancing: the blocks should be evenly distributed among the processors. Given the large number of blocks this can be done efficiently.

3. Implicit and Explicit/Implicit Time-Stepping

We describe a fully implicit time integration scheme designed for block-adaptive grids. The algorithm is built up from well known ingredients (see e.g. the review Knoll & Keyes 2004): three-level second-order implicit time discretization, Newton linearization and Krylov subspace iterative solver using Schwarz type preconditioning. In the design of the implicit time integration scheme we exploit the simplicity of the grid structure in the blocks as well as the natural decomposition of the grid into these blocks: our Schwarz type preconditioner is block-based, which provides a natural, simple and efficient preconditioning. Although implicit schemes are unconditionally stable according to the linear stability analysis, in reality large time steps can induce non-linear instabilities that can crash the code. We describe an adaptive time-step adjusting algorithm that can significantly improve the robustness of the implicit scheme.

We introduce an explicit/implicit time stepping algorithm especially suited for block-adaptive grids. The scheme advances the blocks either with the explicit or the implicit time stepping scheme depending on the local numerical stability.
condition for a given block. Figure 2 shows the distribution of the local stability limits in a two-dimensional cut-plane of a typical magnetosphere simulation. The time step limits range from a hundredth of a second to 60 seconds. If the time step is a few seconds then many blocks are stable even with explicit time stepping. This suggests that the combination of explicit and implicit time stepping can improve the overall efficiency.

The basic concept of the explicit/implicit scheme is very simple. For a given time step, which is limited by accuracy requirements, some parts of the computational domain may allow the inexpensive explicit time integration, while other parts require the more expensive but stable implicit time integration scheme. The blocks of the AMR grid provide a natural way of decomposing the grid into explicit and implicit regions. The algorithm proceeds as follows:

1. Set the time step based on accuracy, efficiency and robustness requirements.
2. Assign blocks to be explicit or implicit based on the local stability condition.
3. Load balance explicit and implicit blocks.
4. Advance explicit blocks in time.
5. Update ghost cells for implicit blocks.
6. Advance implicit blocks in time.
7. Update ghost cells for all blocks.

Load balancing is done by evenly distributing the explicit as well as the implicit blocks among the processors. We use a Peano–Hilbert space filling curve to order the explicit and implicit blocks into two separate ordered lists. Both lists are split into as many equal pieces as the number of processors. For any two
processors the number of explicit (and implicit) blocks can differ by at most one. Using the Peano–Hilbert ordering ensures good data locality, which means that the inter-processor communication is minimized. Although doing load balancing every time step looks expensive, in practice it turns out to require a relatively small fraction of the total execution time. In a typical application the blocks do not change from explicit to implicit or implicit to explicit too often. Even when they change, only a few blocks need to be moved around.

An interesting aspect of the explicit/implicit scheme is the optimal choice of the time step. For an explicit scheme the optimal time step is the largest $\Delta t$ allowed by the numerical stability. For a fully implicit scheme the optimal time step is the largest $\Delta t$ allowed by the accuracy requirements and non-linear stability conditions. In case of the explicit/implicit scheme the time step is limited by accuracy and non-linear stability requirements, but the largest allowed time step may not be the optimal one for speed. This is quite easy to see: the larger the time step is, the more blocks require implicit time stepping; this requires more computation (one needs to solve a larger linear system), which can reduce the performance. The optimal time step is application dependent, and usually there is a relatively flat minimum in the time step/performance curve.

With only a modest investment in coding, the explicit/implicit scheme can be significantly more efficient than the fully implicit scheme. By using the implicit scheme on a subset of the blocks, both the CPU and memory costs are significantly reduced. In addition, the upwind weighted explicit scheme can produce a more accurate solution for fast (close to one cell per time step) moving discontinuities and sharp gradients than the implicit scheme, thus accuracy is significantly improved if the explicit scheme is applied in the regions where fast moving discontinuities and sharp gradients occur.

### 3.1. BATSRUS and SWMF

The implicit and explicit/implicit schemes have been implemented in the BATSRUS code (Powell et. al. 1999) which solves the MHD equations with second-order shock-capturing total variation diminishing (TVD) schemes on a three-dimensional block-adaptive grid. The grid can be Cartesian or one can employ generalized (including spherical and cylindrical) coordinates. The code can solve several extensions of the MHD equations: Hall MHD, multi-species chemistry, multi-fluid MHD, and semi-relativistic MHD. The magnetic field can be split into an analytic and numerical part ($B = B_0 + B_1$) which is very useful in modeling the magnetosphere or the solar corona, where the magnetic field is dominated by a strong but relatively stationary intrinsic field. Although the BATSRUS code is mostly applied to space physics problems (comets, moons, planets, Sun, inner heliosphere, outer heliosphere) it is a general MHD code that can be (and has been) used in other areas as well, e.g. astrophysics or laboratory plasma physics.

While BATSRUS is a very powerful stand-alone MHD code, it becomes even more versatile as part of the Space Weather Modeling Framework, where it is used to model three physics domains: the solar corona, the heliosphere and the magnetosphere. The SWMF with all its components (including BATSRUS) is freely available at http://csem.engin.umich.edu/swmf via registration.
The explicit schemes in the BATSRUS code achieve excellent scaling on distributed memory parallel super-computers and it has been a high priority to get a good parallel scaling for the implicit scheme as well. To the best of our knowledge BATSRUS is the first fully implicit parallel MHD code that works on an AMR grid. Among our applications the implicit time stepping is found to be the most valuable for time-accurate integrations of the MHD equations. This is different from the more typical applications of implicit schemes, where time-marching towards a steady state is considered. Our choices among the dozens of options in the NKS algorithm are geared towards the time-accurate application, and are different from the choices optimal for steady-state problems. The explicit/implicit scheme in BATSRUS has already been successfully used to model the magnetospheres of Earth, Saturn and Uranus (Tóth et. al. 2007; Hansen et. al. 2005; Tóth et. al. 2004).

4. Newton–Krylov–Schwarz Algorithm

The spatially discretized form of the partial differential equations can be written as the system of differential equations $\frac{\partial \mathbf{U}}{\partial t} = \mathbf{R}(\mathbf{U})$, where $t$ is time, $\mathbf{U}$ is the vector of the spatially discretized dependent variables for all the grid cells, and $\mathbf{R}$ is a non-linear function of $\mathbf{U}$. The right hand side may also depend explicitly on time (not shown). Note that in this semi-discretized form the spatial derivatives of $\mathbf{U}$ are approximated with algebraic functions of $\mathbf{U}$.

The implicit time discretization uses a three-level scheme

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

where $\beta = (\Delta t_n + \Delta t_{n-1})/(2\Delta t_n + \Delta t_{n-1})$. For constant time steps $\beta = 2/3$ and the equation above simplifies to the second-order Backward Differentiation Formula (BDF2, Hairer et. al. 1987).

Equation (1) is a large system of non-linear equations for the unknowns $\mathbf{U}^{n+1}$. Instead of solving this non-linear system, we linearize it by approximating $\mathbf{R}(\mathbf{U}^{n+1})$ as

$$\mathbf{R}(\mathbf{U}^{n+1}) = \mathbf{R}(\mathbf{U}^n) + \frac{\partial \mathbf{R}}{\partial \mathbf{U}} \cdot (\mathbf{U}^{n+1} - \mathbf{U}^n) + \mathcal{O}(\Delta t^2) \quad (2)$$

which can be substituted into (1) to arrive at

$$\left[ I - \Delta t_n \beta \frac{\partial \mathbf{R}}{\partial \mathbf{U}} \right] \cdot (\mathbf{U}^{n+1} - \mathbf{U}^n) = \Delta t_n \left[ \beta \mathbf{R}(\mathbf{U}^n) + (1 - \beta) \frac{\mathbf{U}^n - \mathbf{U}^{n-1}}{\Delta t_{n-1}} \right] \quad (3)$$

where $I$ is the identity matrix and the $\mathcal{O}(\Delta t^3)$ terms are dropped. Note that if $\mathbf{R}$ depends explicitly on time (e.g. at the boundaries), then $\partial \mathbf{R}/\partial \mathbf{U}$ and $\mathbf{R}(\mathbf{U}^n)$ should be taken with arguments $\mathbf{U}^n$ and $t_{n+1}$. This linearization can also be regarded as the first Newton iteration towards the solution of the non-linear system (1). Although we have implemented the full Newton iteration, in our unsteady applications it is found to be less efficient than solving the linearized equation (3), or in other words, using a single Newton iteration.
The linearized system is a second-order accurate approximation of the non-linear system, which itself is a second order approximation of the analytic solution at time level \(n+1\). The solutions of the non-linear and the linearized systems have both \(O(\Delta t^2)\) errors (in addition to the truncation errors of the spatial discretization), therefore there is no reason to assume that the solution of the non-linear system is more accurate.

The linear system (3) can be written in a compact form as \(A \cdot x = b\), where the matrix \(A = (I - \Delta t n \beta \partial R / \partial U)\), the unknown \(x = (U^{n+1} - U^n)\) and \(b\) is the right hand side of (3). This linear system is solved with a preconditioned Krylov subspace solver such as GMRES (Saad & Schultz 1986) or BiCGSTAB (van der Vorst 1992), which are well suited for parallelization. We found that while the BiCGSTAB iterative scheme requires much less memory than GMRES, in our most challenging applications the GMRES scheme is more robust and efficient.

Allowing for both left hand and right hand side preconditioning, the matrix \(A\) and the right hand side \(b\) are converted to \(A'\) and \(b'\), respectively, while the solution \(x'\) of the preconditioned equation \(A' \cdot x' = b'\) is converted back to \(x\):

\[
\begin{align*}
A' &= P_L \cdot A \cdot P_R \\
b' &= P_L \cdot b \\
x &= P_R \cdot x'
\end{align*}
\]

where \(P_L\) and \(P_R\) are the left and right preconditioning matrices, respectively.

In the Krylov subspace schemes the matrix \(A'\) is not needed explicitly, only its action on a vector is required. Due to the complexity of the block-adaptive grid structure, calculating, storing and multiplying with the non-zero matrix elements of \(A'\) would be quite complicated. Therefore the matrix-vector multiplication with \(A' = P_L \cdot A \cdot P_R\) is done in three stages. First the vector is multiplied by \(P_R\), which can be done efficiently provided that the preconditioner matrix is simple. Next we use a Jacobian-free evaluation for the multiplication with \(A\), which exploits the fact that \(A\) is composed of an identity matrix and a Jacobian:

\[
\begin{align*}
\left[ I - \Delta t n \beta \frac{\partial R}{\partial U} \right] \Delta U &= \Delta U - \Delta t n \beta \frac{R(U^n + \epsilon \Delta U) - R(U^n)}{\epsilon} + O(\epsilon)
\end{align*}
\]

where \(\Delta U\) is an arbitrary vector and \(\epsilon\) is a small parameter. Finally we multiply with the matrix \(P_L\), which can be done efficiently if \(P_L\) is a well chosen preconditioner. The \(\epsilon\) parameter is typically taken to be around the square root of the round off errors. For double precision arithmetic we use \(\epsilon = 10^{-6}\).

The Jacobian-free evaluation offers a lot of flexibility in terms of the discretization of \(A\). We find that in our applications it is not necessary or beneficial to use a spatially second-order discretization for \(A\). It can be shown (see e.g. Keppens et. al. 1999) that the Jacobian matrix \(\partial R / \partial U\) may be evaluated from a first order spatial discretization, still the whole scheme remains second-order accurate in space and time. As observed by many authors (see Knoll & Keyes 2004, and references therein) it is advantageous to use a first order upwind type spatial discretization for the Jacobian matrix, because it results in a more efficient and more robust scheme than a second-order evaluation of the Jacobian. This behavior is expected, since a first order upwind weighted scheme
has a stronger diagonal dominance than the second-order schemes (which tend to result in an anti-symmetric matrix for hyperbolic systems), and the Krylov subspace methods converge better for a diagonally dominant matrix. Since the preconditioner is also based on a first order discretization, it preconditions more effectively for a first order discretization of $A$. Finally, the first order upwind schemes have a smoother $\mathbf{R}(\mathbf{U})$ function than second order schemes applying non-smooth limiters.

For time-accurate calculations it is neither necessary nor efficient to solve the linearized equations to machine accuracy. In our typical applications it is sufficient to reduce the L2 norm of the residual by a factor of 1000. We have experimented with varying the tolerance (which is an adjustable parameter), and found that requiring a smaller residual increases the number of Krylov iterations without significantly changing the solution as compared to the truncation errors at a practically achievable grid resolution. Using a more relaxed tolerance led to accuracy and robustness problems. The optimal choice of the tolerance is problem dependent. With good preconditioning a factor of 1000 reduction requires typically 10 to 30 Krylov iterations. Note that the errors in the different variables can only be added up after normalizing the components of $\mathbf{U}$ individually.

Most preconditioners require the calculation of the matrix elements of some approximations of $A$. For sake of parallel efficiency we use a Schwarz type preconditioner, using only local data that is available on the processor without communication. The block structure of the grid offers a natural choice: each implicitly integrated block is preconditioned individually. This choice has two advantages: (i) The preconditioner does not depend on the parallel distribution of data, since the data in a grid block are guaranteed to be on the same processor. (ii) The simple structure of the grid block results in a simple matrix, which can be easily and efficiently preconditioned. While including relatively limited information in the preconditioner may reduce the convergence rate of the Krylov solver, using a simple and fully parallel preconditioner reduces the time spent on calculating and applying the preconditioner matrices.

The preconditioner matrices are based on the Jacobian matrix $A$ restricted to an individual grid block, which is denoted by $\bar{A}$. Using a first order discretization for $\bar{A}$ results in a block-heptadiagonal matrix. The matrix elements of $\bar{A}$ can be obtained in various ways. We use an approach (see Keppens et. al. 1999), in which the matrix elements are calculated by analytic differentiation of the discretized equations at the level of fluxes and source terms, but the derivatives of the fluxes and source terms are calculated by numerical differentiation. Finally the preconditioning matrices $P_L$ and/or $P_R$ are constructed from the relaxed modified block incomplete lower upper (MBILU) Axelsson & Lindskog (1986) factorization of $A$.

5. Time-Step Adjustment

While the linear stability analysis suggests that the implicit time-stepping scheme is unconditionally stable, the nonlinearity of the equations can result in nonlinear instability if the time step is increased too much. We have designed
a time-step adjustment algorithm to maintain the time step as large as possible without crashing the code.

For the MHD equations the non-linear instability usually manifests itself as negative thermal pressure that is calculated from the total energy density by subtracting the kinetic and magnetic energies. If thermal energy is a small fraction of the total energy, numerical errors can easily result in negative pressure. Occasionally the density may also become negative due to truncation errors. We use the minimum (over all grid cells) of the relative change in pressure $p$ and density $\rho$

$$Q = \min_i \left( \frac{p_i^{n+1}}{p_i^n}, \frac{\rho_i^{n+1}}{\rho_i^n} \right)$$

as an indicator for stability and adjust the time step according to the following algorithm:

1. If $Q < 0.3$ then redo time step with $\Delta t'_n = \Delta t_n/2$
2. If $0.3 \leq Q < 0.6$ then reduce the next time step to $\Delta t_{n+1} = 0.9\Delta t_n$
3. If $Q > 0.8$ then increase the next time step to $\Delta t_{n+1} = \min(\Delta t_{\text{max}}, 1.05\Delta t)$
4. Otherwise use $\Delta t_{n+1} = \Delta t_n$.

If the first case is invoked, the algorithm is applied recursively, however the next time step is only allowed to be increased if there was no prior reduction in the current time step.

This algorithm is obviously not unique but it may be easily modified and adapted. In our most challenging applications this algorithm has greatly improved the robustness of the implicit time integration scheme.
6. Conclusions

As a demonstration of the efficiency of the explicit/implicit time integration scheme, we present scaling results on an SGI Altix super computer (Columbia). Figure 3 shows scaling curves for three Hall MHD magnetospheric simulations using explicit, implicit and explicit/implicit time stepping, respectively. This is strong scaling with a fixed problem size: the grid consists of 2.5 million cells ranging from $8 R_E$ to $1/16 R_E$. The figure demonstrates the near perfect scaling of the explicit scheme. The fully implicit and explicit/implicit algorithms are about 14 and 20 times faster on 32 CPU-s, respectively, and they still scale quite well up to 256 processors. For Hall MHD most of the blocks require implicit time steps, so there is relatively little difference between the efficiency of the implicit and explicit/implicit algorithms. In our typical classical MHD applications there is a factor of 2 to 4 improvement.

Acknowledgments. This work has been supported by NASA AISRP grant NNG04GP89G, NASA ESS CT cooperative agreement NCC5-614 and by DoD MURI grant F49620-01-1-0359.

References