Computer Physics Communications 327 (2026) 110293

Contents lists available at ScienceDirect

Computer Physics Communications

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

Computational Physics

Fast solvers for tokamak fluid models with PETSc

Mark F. Adams [a,*], Jin Chen [b], Benjamin Sturdevant [c]

[a] Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[b] Princeton Plasma Physics Laboratory, Princeton, NJ, USA
[c] Rensselaer Polytechnic Institute, SCOREC, Troy, NY, USA

ARTICLE INFO

The review of this paper was arranged by Prof. Andrew Hazel

Keywords:
Multigrid
MHD solvers
SPARC

ABSTRACT

Multigrid (MG) is widely recognized as a highly effective solver for the model problem, the Laplacian, but textbook MG fails on most problems of interest. MG methods have been applied to complex, real-world applications with careful consideration of the physical model and discretization. This work develops the first step in applying MG methods to science and engineering relevant magnetohydrodynamics (MHD) tokamak models in the *M3D-C1* (https://m3dc1.pppl.gov) fusion energy science code. The semi-implicit time integrator in *M3D-C1* is composed of many linear solves. The implicit advance of the momentum equation is the most challenging and is the focus of this work.

The current production solver in *M3D-C1* is a block Jacobi (BJ) preconditioner within a Krylov solver, where blocks group degrees of freedom on planes of constant toroidal coordinate. BJ convergence degrades as the number of planes increases due to the spectral properties of the matrix preconditioned with BJ. The partially magnetic field-aligned, regular toroidal grid structure in *M3D-C1* is amenable to semi-coarsening geometric MG in the toroidal direction. This paper develops such a solver and demonstrates competitive performance on a runaway electron model of a SPARC (https://cfs.energy/technology/sparc) disruption, and superior robustness on a stellarator model on which the BJ solver fails to converge.

1. Introduction

M3D-C1 is an extended magnetohydrodynamic (MHD) code used for modeling magnetized plasma equilibrium, stability, and nonlinear dynamics, primarily in toroidal fusion devices. The governing equations treat the plasma as a coupled system of electrically conducting ion and electron fluids, leading to a complex, coupled nonlinear PDE system defined on nontrivial geometries. The code is actively used to address critical challenges in tokamak plasmas, including large-scale instabilities that degrade thermal confinement and disruptions that result in complete energy loss and may cause damage to reactor-scale tokamaks. The *M3D-C1* code follows design ideas that originated in the earlier *M3D* code but is developed independently. Its name refers to the use of C^1-continuous finite element spaces, which enforce continuity of both solution fields and first derivatives across element boundaries. To enable efficient solution of the governing equations across a wide range of spatial and temporal scales, *M3D-C1* employs advanced numerical methods, including high-order finite element discretizations on unstructured meshes and both fully implicit and semi-implicit time integration schemes. Scalability is obtained through domain decomposition and parallel sparse linear algebra solvers. *M3D-C1* is used by over a dozen physics groups around the world including university, national laboratory, and industry users.

Impact: The advanced time integration scheme in *M3D-C1* requires solves of roughly 10 separate linear systems per time step depending on the model. The base equations are given in Appendix A and further details can be found in [1]. These solves lie on a broad spectrum of difficulty, with the momentum solve being the most challenging and least robust. This work focuses on the momentum solve and only on the PETSc solver component (e.g., not matrix assembly). The solver developed here is, however, fully applicable to "hard" solves in *M3D-C1*. Many of the solves are mass dominated and easy to solve. The momentum solve accounts for about 3% of the total run time, but is the most expensive, least robust and the starting point for developing fast solvers in *M3D-C1*.

Multigrid (MG) methods solve a fine-grid problem by leveraging a hierarchy of coarse resolution grids. On each grid, coarse function spaces are constructed, and residuals and coarse grid corrections are transferred between levels via restriction and interpolation operators. At each level, inexpensive solvers, known as *smoothers*, are used to reduce high-frequency error components, on that grid, and the coarse grid solver must eliminate all error. MG can provably solve the Laplacian, *the model*

* Corresponding author.
E-mail address: mfadams@lbl.gov (M.F. Adams).

https://doi.org/10.1016/j.cpc.2026.110293
Received 11 December 2025; Received in revised form 21 May 2026; Accepted 27 June 2026
Available online 8 July 2026
0010-4655/Published by Elsevier B.V.

problem, to discretization error accuracy with a work complexity of about five residual calculations, using the full MG cycle [2]. Achieving this so-called *textbook multigrid efficiency* (TME) on a given problem requires understanding and exploiting the structure of the equations and discretizations. A variety of multigrid methods have been developed [2] and TME has been achieved to some extent on MHD problems [3] and CFD problems [4].

The dynamics of tokamak fusion plasmas are highly anisotropic, dominated by magnetic guide fields that confine the charged particles to trajectories around the torus (Fig. 1). All tokamak models exploit this structure with a *poloidal plane* grid that is approximately perpendicular to the magnetic guide field. These $3D$ domains are invariably tensor products of $2D$ poloidal plane grids (and discretizations) with a $1D$ toroidal grid. Many tokamak codes use unstructured grids in the poloidal plane that are thus "extruded" around the torus. The structured grid in the toroidal direction suggests the use of a standard approach to anisotropy on structured grids [2], geometric semi-coarsening, which is the focus of this paper. Applying MG to the poloidal plane is the subject of future work.

The $3D$ velocity field in M3D-C1 is decomposed into three scalar functions corresponding to three waves of the MHD system (Appendix A.3). This model can include one, two, or all three waves. Using all three is common in practice, but reduced systems are useful for verification. With this decomposition, on a Cartesian grid without nonlinear terms, the three scalar equations decouple. In practice the variables are weakly coupled, see Appendix C for quantitative examples. Because of this weak coupling, *FieldSplit* preconditioners in *PETSc* that treat each variable separately in a (3) block Gauss-Seidel iteration are worth investigating. *FieldSplit* preconditioners support custom blocks for each wave, which could be effective given that these waves have different anisotropies. This is a subject of future work.

These multigrid ideas are fairly generic and some of them are applicable to other production MHD tokamak codes such as *NIMROD*,[1] which uses a different formulation of the extended MHD system that also exploits the toroidal structure of tokamaks, but with a spectral decomposition in the toroidal coordinate. The resulting matrix is also a block diagonal matrix under ideal, linear conditions, but the number of blocks is much larger than three, being the number of toroidal modes kept in the system, on the order of $30-50$. Thus *NIMROD* exploits the structure that we exploit with $1D$ multigrid at the formulation level. The resulting $2D$ linear system is similar to some extent to the poloidal plane solve in *M3D-C1* in that the matrix is likely nearly block diagonal and both use high-order accurate finite element discretizations of unstructured $2D$ poloidal grids. This structural similarity suggests that the two future work topics in this paper, geometric MG in the poloidal plane and custom small block Jacobi smoothers, could be broadly applicable to *NIMROD*.

The main contribution of this paper is demonstrating that multigrid methods can provide a path toward faster, scalable, and robust solvers for even the most complex and ill-conditioned of science and engineering relevant magnetized plasma models and that semi-coarsening in toroidal coordinates is a useful first step in this process for common tokamak codes. This work proceeds by describing the *M3D-C1* velocity evolution equations and solver methods in Section 2, the model problems for performance evaluation and verification are defined in Section 3, performance results are presented in Sections 4 and 5 concludes.

2. *M3D-C1* equations, multigrid and solver structure

Multigrid design methodology: While iterative solvers have the potential to be effective PDE solvers, they require an understanding of the equations – and their discretizations – to achieve this potential [3–5]. Given the equations and discretizations, analysis techniques are available to guide the design of, and even prove the optimality of, multigrid methods. Examples of such tools are local Fourier analysis (p. 98 [2]), h-ellipticity (p. 121 [2]), principal component analysis, which in this context refers to writing the equations in matrix form, taking the determinant, and ordering the terms to understand the dominant terms of the system (e.g., whether it is elliptic).

2.1. *M3D-C1* equations

Expressing the explicit algebraic form of the *M3D-C1* model for analysis is not feasible and the methods used here are generic for elliptic operators. *M3D-C1* uses $2D$ unstructured grids in the poloidal plane that are extruded in the toroidal direction. These prism elements use a $2D$ C^1-continuous Bell element in the poloidal plane [6], together with cubic Hermite elements in the toroidal direction [7]. These discretizations include not only the variable itself as a degree of freedom (DoF), the zeroth derivative, but also higher derivatives for a total of 12 equations per physical DoF.

Parabolization is a transformation of the extended MHD equations in *M3D-C1* (Appendix A.2) [8–11]. The momentum equation is parabolized by taking a time derivative and then eliminating the resulting time derivatives of density and magnetic field using the ideal MHD equations. This results in two fourth order, symmetric, curl operators (U, χ) and one Laplacian (ω), discretized with high order, C^1-continuous finite elements. Symmetry and ellipticity are not necessary to successfully apply multigrid to resistive MHD [3], but they are very useful properties and critical for solving these problems with multigrid.

2.2. *M3D-C1* solvers

M3D-C1 solvers use *PETSc*[2] Krylov solvers (FGMRES or GMRES) and employ a block Jacobi (BJ) preconditioner, with exact subdomain solves, in which each block corresponds to a poloidal plane. This is a natural decomposition of the problem given that an unstructured grid is used in the poloidal plane to capture the geometry of the tokamak wall and magnetic flux surfaces. This $2D$ grid is extruded with a regular $1D$ grid in the toroidal direction. The structured toroidal grid makes semi-coarsening geometric MG particularly simple, with geometric 2:1 coarsening and a 3-point stencil[3] prolongation operator P. Having the application compute Jacobians on coarse grids is not practical, but given P the coarse grid operators can be constructed algebraically with a Galerkin process: $A_{i+1} = P^T A_i P$. This results in a geometric/algebraic multigrid solver with a purely algebraic interface [12].

The PETSc solver interface in M3D-C1 takes matrix and vector objects, populated by the application in an object-oriented interface. The numerics are performed with high level controls on the host and distributed linear algebra back-ends, some of which are third-party libraries like parallel direct solvers *MUMPS* and *SuperLU*, and device linear algebra libraries like *cuSPARSE*, as well as non-numerical methods such as mesh partitioners for load balancing. Stand-alone PETSc has built-in host linear algebra and geometric and algebraic MG preconditioners. All data can remain device resident with time integrators, nonlinear solvers and Krylov linear solvers operating through abstract interfaces to the linear algebra primitives that run on the device. A solver object is then created, the solver is called, and the solution is extracted from the PETSc solution vector. Solver parameters can be set in the code, but using command-line parameters is recommended for runtime flexibility (see Appendix B for an example).

M3D-C1 computes one new Jacobian matrix every time step, without a nonlinear iteration. Furthermore, the preconditioner is lagged and with a *refresh period* provided by the user because the setup costs of matrix factorizations, etc., are significant. This *setup*, or refresh, for the

[1] https://nimrodteam.org

[2] https://petsc.org

[3] $\left]\frac{1}{2}\quad 1\quad \frac{1}{2}\right[$

production BJ solver is primarily full LU factorizations on each plane. The MG solver also requires a full plane LU factorization for the coarse grid and all grids when the BJ solver is used as the smoother. Additionally, MG has a cost in the Galerkin coarse grid construction. As the physics evolves, solver convergence degrades from the lack of consistency between the solver and matrix. This refresh model is very simple and a smarter algorithm could clearly be useful in these problems where the physics appears to be reaching a quasi steady state and the refresh period could be increased dramatically.

Solver tolerance. The relative solver tolerance, $rtol = \frac{|b-A\tilde{u}|}{|b|}$, is the primary parameter that governs solver accuracy and cost. The *M3D-C1* team has determined that an $rtol = 10^{-9}$ is reliable and is used in production. For the purpose of experiments, we wish to optimize the solver by reducing the tolerance by studying accuracy in a way that would not be practical for users. *M3D-C1* reports nine quantities (kinetic and magnetic energies, etc.) at each of the 100 time steps in the test problem. The verification test compares this data to data collected with a high-accuracy solver tolerance. The test is deemed to pass if each value, x, and its counterpart in the high-accuracy data, $\hat{x}$, has a relative difference $\frac{|x-\hat{x}|}{|\hat{x}|}$ less than 10^{-3}. We modified this test to filter out false positives from $\hat{x}$ values that are near zero by first scanning for the largest absolute value for each of the nine quantities for use as $\hat{x}$. The failure tolerance is reduced to 10^{-2} and the number of tests that violated the 10^{-3} tolerance is reported. This test is used to set custom tolerances for the SPARC problem in Section 4.1. Data with the production tolerance of $rtol = 10^{-9}$ is reported in Sections 4.2 and 4.3.

2.3. Multigrid background

Grids are traversed according to various cycling strategies in multigrid, the most common being the *V-cycle*, in which the coarsest-grid solve forms the base of the "V". Smoothing steps descending the hierarchy (on the left leg of the "V") are called *pre-smoothing*, while those applied on the ascent (the right leg) are called *post-smoothing*. A common shorthand for the number of pre-smoothing and post-smoothing steps is $V(pre, post)$, e.g., $V(1,1)$ uses a single pre-smoothing and post-smoothing step. Two smoothers are used in this work: point-block Jacobi (PBJ) smoothers and the BJ solver can be repurposed as a smoother. With C^1 elements, all degrees of freedom are on the vertices of the mesh. The Bell elements are about fifth-order accurate (in plane) and the cubic splines are fourth-order accurate [7]. This discretization has 12 DoFs per vertex, per physical zeroth derivative DoF, and with three DoFs in the momentum solve the matrix has a *block size* of 36, that is 36 equations per vertex. PBJ smoothers in PETSc compute explicit, dense inverses of these diagonal blocks, which can be applied effectively on GPUs with dense matrix-vector products. A BJ smoother is a powerful and expensive smoother and is used in a $V(0,1)$-cycle for the most challenging problems investigated here. The PBJ smoother is fast and uses less memory and is useful on some of the problems.

Previous work on MHD solvers. While Adler et al. have developed multigrid methods for a two field viscoresistive MHD model with curl operators [13–15], and Chacón used a parabolization (or Schur complement) approach to apply multigrid to Laplacians in an MHD solver [5], there is no prior work on applying multigrid to solve systems approaching the complexity of those in *M3D-C1*.

3. Model test problems

This section describes test problems used for performance evaluation. Two tests are used: a runaway electron model of a SPARC tokamak disruption, and a stellarator problem. The stellarator test uses a coordinate transformation in *M3D-C1* for the geometry of a stellarator. Fig. 1 shows the geometry of a generic tokamak (left) and a generic stellarator (right).

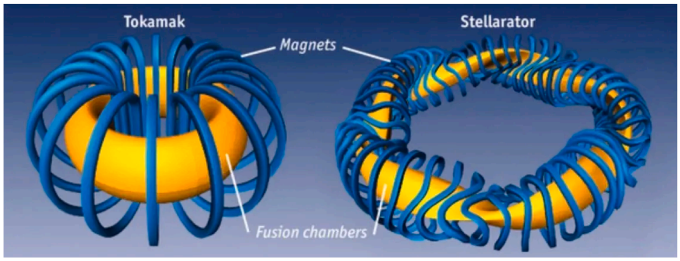


Fig. 1. Traditional tokamak and stellarator geometry.

3.1. SPARC disruption with runaway electrons problem

SPARC is a compact, high-field ($B_0 = 12.2\,\mathrm{T}$), high-current ($I_p = 8.7\,\mathrm{MA}$) tokamak designed to achieve $Q > 2$ in deuterium–tritium plasmas. *M3D-C1* incorporates a fluid runaway electron model coupled with the bulk MHD equations [16,17].

3.2. Quasi-axisymmetric stellarator test

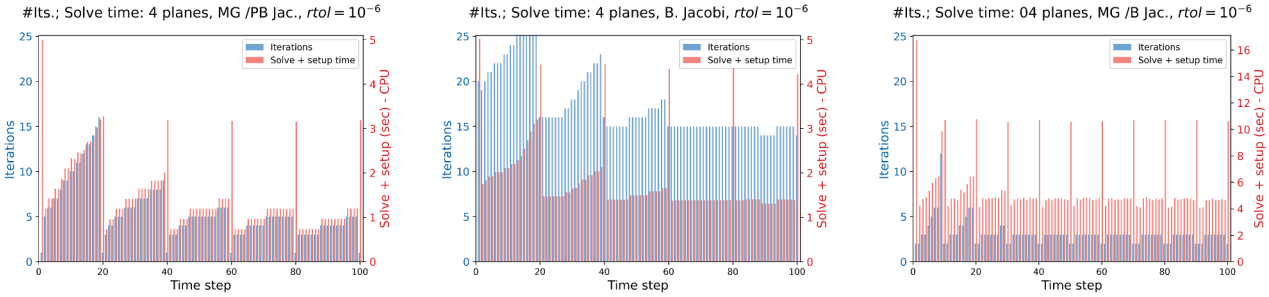
The second test is a stellarator problem using M3D-C1's recently developed coordinate transformation to adapt to non-axisymmetric stellarator geometry [18]. The configuration is a quasi-axisymmetric (QA) stellarator optimized for low quasisymmetry error, good energetic particle confinement, and self-consistent bootstrap current [19,20], with a minor radius of 1.70 m and volume-averaged $B = 5.86\,\mathrm{T}$. Bootstrap current, which represents a significant fraction of the total current density in QA stellarators, is included self-consistently via the Sauter–Redl–Landreman formulation [20–23], which exploits the isomorphism between axisymmetric and quasi-symmetric geometries. This test case is significantly more challenging for the solver than the SPARC problem: the BJ solver fails to converge (Fig. 10, whereas the multigrid solver remains effective, albeit using BJ as the smoother, demonstrating the robustness advantage of the MG approach on non-axisymmetric configurations. This project is under active development and multiple physicists have observed the BJ solver fail and used the MG solver to continue work.

4. Performance evaluation

This section evaluates the performance of the solvers developed in this work on these two fusion modeling problems. The Perlmutter machine at NERSC is used for all numerical studies. Perlmutter has CPU nodes with two 64-core AMD EPYC 7763 processors and GPU nodes with one 64-core AMD EPYC 7763 and 4 NVIDIA A100 GPUs. The 2D poloidal plane mesh is partitioned into processor subdomains in an offline setup phase. This mesh is extruded around the tokamak or stellarator with a given number of planes. The MUMPS LU solver is used for all direct solves.

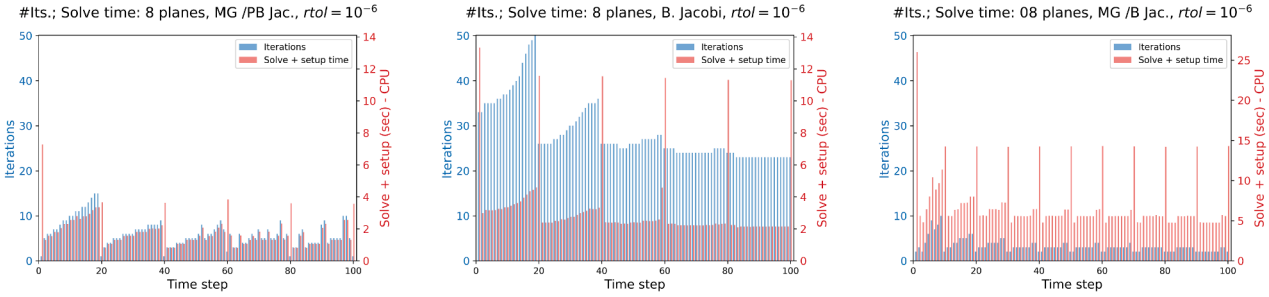
4.1. SPARC problem with custom solver tolerances

The SPARC test problem is run with $4 - 64$ planes in a scaled speedup study using the custom solver tolerance of $rtol = 10^{-6}$ and 10^{-7}. Each plane contains 14,200 vertices with 36 DoFs per vertex, which results in 511,200 equations per plane. The 2D unstructured mesh of the poloidal plane is partitioned into 32 processor subdomains, resulting in 15,975 DoFs per process and about 12,000,000 non-zeros per process. All tests use 16 processes per CPU socket, which drive 4 GPUs in the GPU tests. For the CPU study, one plane is placed on each node with two CPU sockets, and for the GPU study two nodes with one CPU socket each are used for each plane, resulting in 8 GPUs per plane and about 64K equations, or 48M non-zeros, per GPU.



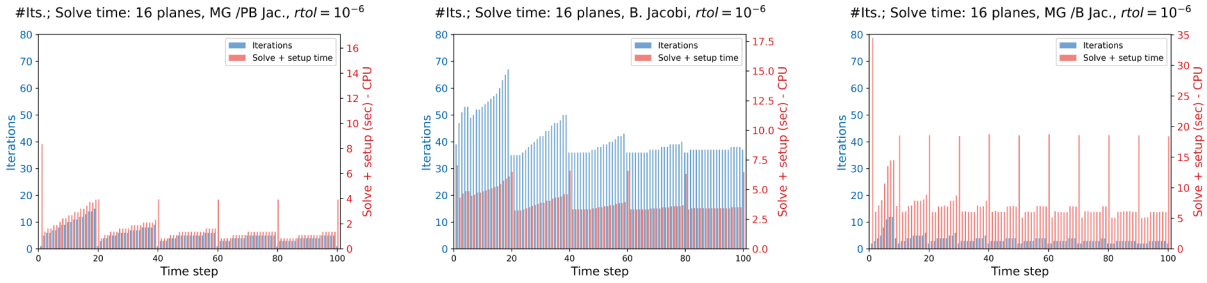
(a) MG $V(1,1)$ PBJ smoother (b) Block Jacobi (c) MG $V(0,1)$ Block Jacobi

Fig. 2. 4 plane case, iteration counts (left axis/bar) and solve times with setup (right axis/bar).



(a) MG $V(1,1)$ PBJ smoother (b) Block Jacobi (c) MG $V(0,1)$ Block Jacobi

Fig. 3. 8 plane case, iteration counts (left/blue), solve times with setup (right/red) vs time step. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



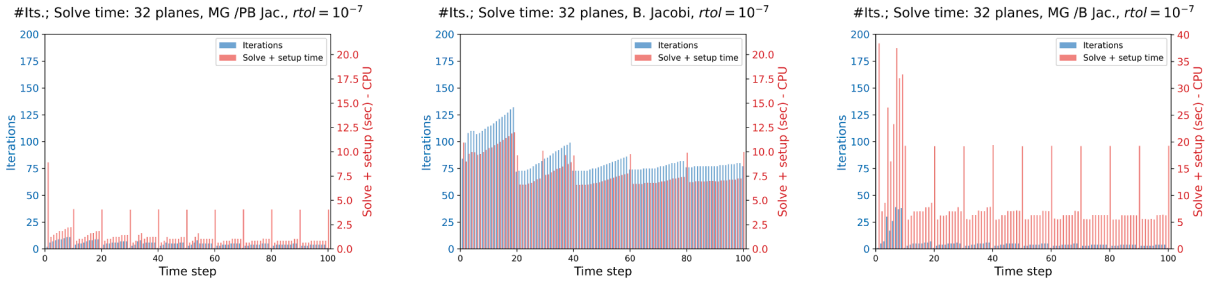
(a) MG $V(1,1)$ PBJ smoother (b) Block Jacobi (c) MG $V(0,1)$ Block Jacobi

Fig. 4. 16 planes case, iteration counts (left/blue), solve times with setup (right/red) vs time step. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The custom tolerance of $rtol = 10^{-6}$ in the 4, 8 and 16 plane cases, and $rtol = 10^{-7}$ in the 32 and 64 plane cases is used. There are 11 and 10 warnings with PBJ smoothing and BJ solver, respectively, in the 64 plane case where the 10^{-3} tolerance in the verification test (Section 2.2) was violated. These warnings occurred in time steps $2-13$, the early growth phase of the test. Note, this result is interesting in that it indicates that the accuracy at a given solver tolerance is similar, which given the different spectral properties of MG and BJ (MG damps the spectra of the error evenly in theory). The time step for the 64 plane case is one half that of the other tests, due to CFL constraints. A refresh period of 20 is used for 4, 8 and 16 plane cases, and a period of 10 is used for the 32 plane case. In the 64 plane case the MG solver with PBJ smoother was refreshed every time step and the BJ solver and MG solver with BJ smoother were refreshed every 10 time steps. The $V(1,1)$-cycle PBJ smoother performance was poor in the 64 plane case, which motivated increasing the smoothing by $6x$, which reduces GMRES restarts and improved performance significantly. The source of this increase in difficulty is not well understood, but it is likely a new nonlinear mode is resolved with increased resolution.

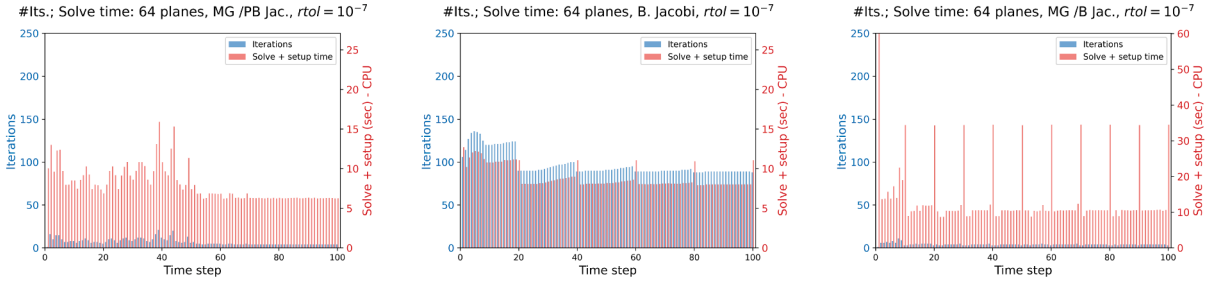
Figs. 2–6 show the iteration count (left axis) and solve time (right axis) that includes the periodic solver setup, for each time step as reported by the *M3D-C1* timers. Each case is run on CPU nodes with the MG $V(1,1)$-cycle solver with PBJ smoothing, with the production block Jacobi solver, and with the MG $V(0,1)$-cycle with the BJ smoother. Note, one iteration of GMRES preconditioned with PBJ as the pre-smoother is used with BJ post-smoothing and is denoted as a $V(0,1)$-cycle for clarity.

The $V(0,1)$-cycle with BJ smoothing is not competitive (note the different time scales) and is more sensitive to inconsistency with the operator. In the 32 plane case in Fig. 5c the $V(0,1)$-cycle takes more iterations than the $V(1,1)$-cycle with PBJ smoothing even though it is a uniformly more powerful (expensive) solver. This is not well understood but it can be attributed to the inconsistent approximation "shooting" the solution further in the wrong direction.



(a) MG $V(1,1)$ PBJ smoother (b) Block Jacobi (c) MG $V(0,1)$ Block Jacobi

Fig. 5. 32 plane case, iteration counts (left/blue), solve times with setup (right/red) vs time step. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) MG $V(6,6)$-cycle PBJ refresh=1 (b) Block Jacobi (c) MG $V(0,1)$ Block Jacobi

Fig. 6. 64 plane case, iteration counts (left/blue), solve times with setup (right/red) vs time step. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Total velocity solve times (sec) in the *M3D-C1* solve phase timers, $^*V(6,6)$-cycle.

Number of planes	4	8	16	32	64
Multigrid / PBJ	147	181	176	188	825*
Block Jacobi	181	320	516	1293	1704

Some observations from this data are as follows:

- The MG solver converges quickly when the preconditioner is refreshed or consistent with the operator, converging in one iteration in the $4-16$ plane cases.
- The MG convergence degrades more dramatically than block Jacobi's without a refresh;
- Block Jacobi solver iteration counts increase with number of poloidal planes, as expected;
- The 32 and 64-plane cases are noticeably harder and a fully refreshed MG solve no longer converges in one iteration and a tighter solver tolerance is required to satisfy the custom verification test;
- The 64-plane case is much harder than the 32 plane case, motivating the use of a $V(6,6)$-cycle and a solver refresh every time step;

4.1.1. Scaled speedup data

Table 1 shows the velocity solve stage time, as measured with *M3D-C1* timers, which is a superset of the PETSc's "KSPSolve" solve times shown in Table 2.

The PETSc solve time and total number of solver iterations over the 100 time steps, for both the multigrid / PBJ and block Jacobi solvers, is shown in Table 2.

The good algorithmic scaling of multigrid is demonstrated although the problems are noticeably harder at 32 and 64 planes, where the tolerance is reduced to $rtol = 10^{-7}$, the refresh rate and V-cycle are modified, and a $V(6,6)$-cycle is used to compensate.

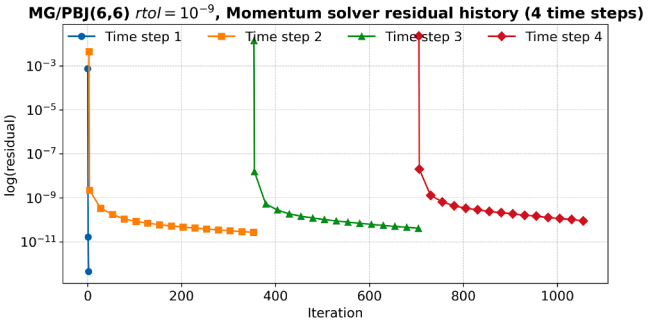


Fig. 7. 64 plane case with V(6,6) PBJ smoother. First three residuals plotted, every 25 thereafter.

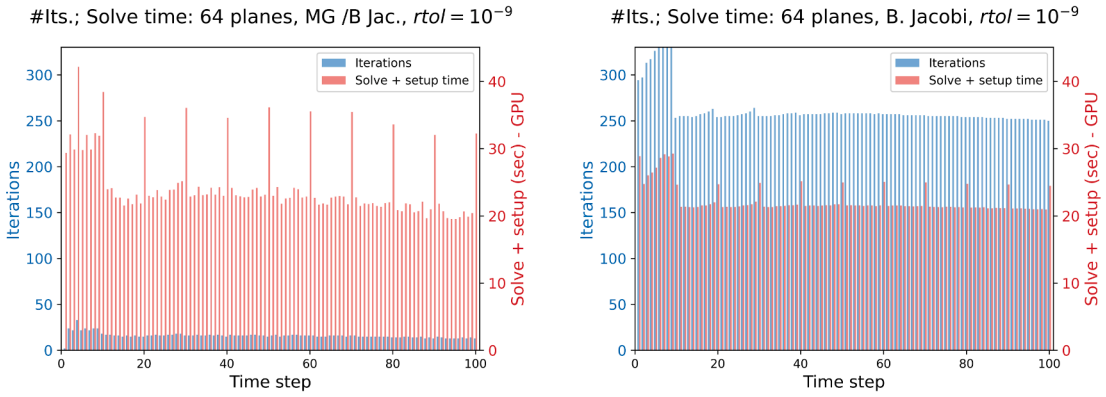
4.2. Performance with standard user tolerance

The last section evaluated the CPU performance of the SPARC test problem with custom solver tolerances. This section uses the high-accuracy tolerances used in practice on GPUs as well as CPUs. We find that the PBJ solvers are not effective and the BJ solver needs to be used as the MG smoother. The standard tolerance used in production is $rtol = 10^{-9}$ to provide a safe solver without the need for careful analysis for optimal parameters. At this high level of accuracy, the PBJ smoother is prone to stagnation and is not viable (see Fig. 7). To address this issue, the block Jacobi solver is re-purposed as the smoother and is used only once per iteration as the post-smoother.

The Perlmutter GPU nodes have one AMD socket as opposed to two for the CPU nodes, which required the use of two GPU nodes, 8 NVIDIA A100 GPUs per plane, to keep the same number of processes and sub-domain size per socket as in the CPU configuration. Fig. 8 shows the solve time and iteration count history for the 64 plane case on GPUs

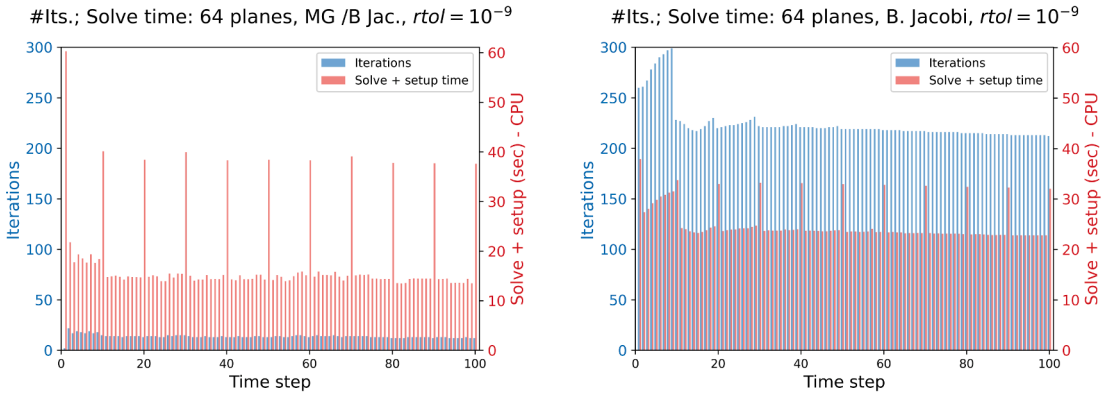
Table 2
Total solver time including setup (sec) [total iterations] PETSc "KSPSolve" time, $^*V(6,6)$-cycle.

# planes	4	8	16	32	64
Multigrid / PBJ	135 [551]	163 [615]	156 [551]	154 [476]	772 [430]*
Block Jacobi	164 [1767]	264 [2848]	498 [5585]	1261 [14,055]	1671 [18,353]



(a) MG $V(0,1)$, block Jacobi smoother (b) Block Jacobi solver

Fig. 8. 64 plane, iterations (left axis/bar), GPU solve times (right axis/bar) vs time step. Solve times include periodic setup.



(a) MG $V(0,1)$, block Jacobi smoother (b) Block Jacobi solver

Fig. 9. 64 plane, iterations (left axis/bar), CPU solve times (right axis/bar) vs time step. Solve times include periodic setup.

Table 3
Total solver time including solver setup (total iterations) in seconds on GPU nodes, $rtol = 10^{-9}$.

# planes	64
Multigrid V(0,1) / BJ	2150 (1653)
block Jacobi (BJ)	2188 (26,203)

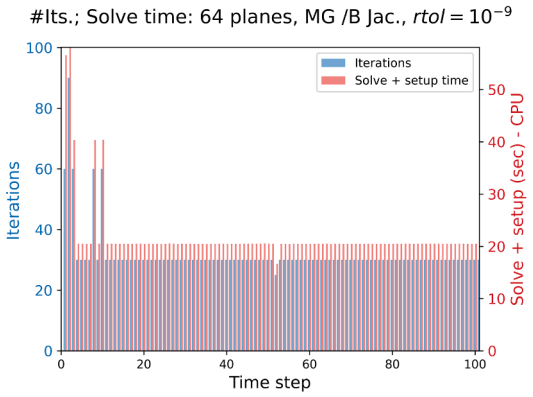
Table 4
Setup cost at refresh periods in seconds (cost of symbolic setup in first time step).

	GPU	CPU
MG $V(0,1)$-cycle	12 (14)	25 (33)
BJ	3 (1)	9 (1)

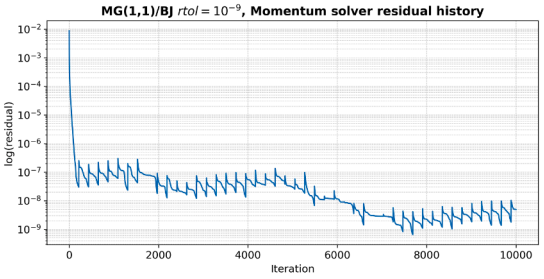
with a solver tolerance of $rtol = 10^{-9}$, and a solver refresh at every 10 time steps. Fig. 9 shows data for this solve on the CPU configuration.

The solve phase performance, the time in the Krylov iterations after the setup, is similar for the CPU and GPU configurations in BJ, which can be attributed to the fact that MUMPS is run on the CPU and calls ScaLAPACK for dense LU solves in the multifrontal algorithm and is therefore not a full GPU solver. The solve phase in LU has data dependencies that inhibit its performance on GPUs, whereas a sparse matrix vector product, the kernel of MG, has few data dependencies. The solve time in the MG solver is significantly faster on the CPUs. We also observe some differences in the iteration counts in both solvers, which is not understood. There are no known non-deterministic operations in this solver, but floating point differences could be the source of this issue – the residual drops quickly in MG and the convergence rate, especially in the early time steps is very slow. The condition number of these matrices is very high and the solver tolerance is very tight, which could explain these results. The solve times for MG and BJ are similar on the GPU. The iteration count with MG is reduced by about $16x$ from that of the BJ solver (Table 3).

Table 3 shows the time in the PETSc solver, with setup costs, as measured with PETSc's "KSPSolve" timer (with the total number of solver iterations over the 100 time steps) for multigrid $V(0,1)$ with the BJ smoother and the BJ solver with the GPU configuration.



(a) MG $V(0,1)$, block Jacobi smoother



(b) Block Jacobi solver residual history of failed solve

Fig. 10. Stellarator 8 plane case: iterations and solve times vs time step (left), stagnated residual history (right) using the BJ solver (jumps result from FGMRES restarts).

Complexity analysis: This data shows that the total solve time is about the same with the multigrid solver and the total iteration count, and hence number of block Jacobi applications on the fine grid, is reduced by a factor of about $16x$. With a $V(0,1)$-cycle, MG applies the block Jacobi solver once per iteration on the fine grid, as well as once on every coarse grid. The work and memory complexity in terms of LU solves is a geometric sum $1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\ldots$ of that of the fine grid, which results in an asymptotic work complexity of $2x$ the BJ solver. This is a lower bound on the cost of the MG solver. The computational depth with seven levels is $7x$ higher in the MG solver, assuming the cost of every grid is the same, which indicates an upper bound of $7x$ more cost per iteration with MG / BJ smoothing. The $16x$ fewer iterations with MG and approximately equal solve times suggests the MG times are about $2x$ higher than the upper bound with this simple model. This indicates that coarse grids are performing poorly, which is the subject of future work.

4.2.1. Setup cost

The refresh costs are not measured separately in the *M3D-C1* timers, but they can be inferred from the jump in the solve time at the refresh period. The cost per iteration can be assumed to be constant. The MG refresh costs are higher than the BJ solver from the Galerkin coarse grid construction and the first setup is larger due to symbolic setup of this Galerkin operator. The MG solver LU factorization costs are higher than the BJ solver because the $V(0,1)$-cycle factors matrices on seven levels instead of one. Table 4 shows the cost of the setup inferred from Figs. 8 and 9.

This data shows a significant speedup with GPUs in the setup phase. This can be attributed to the high arithmetic intensity (flops per byte of data) of both the matrix triple product in the Galerkin coarse grid construction and the LU factorizations.

4.3. Stellarator problem performance

The stellarator simulations are performed on a semi-structured grid of 8 toroidal planes with 1.88×10^5 three-dimensional elements. The stellarator test described in Section 3.2 uses the production solver tolerance $rtol = 10^{-9}$. This test is relatively small with 8 toroidal planes with 1.88×10^5 three-dimensional elements and about 3.4×10^6 equations. The focus in this experiment is on the convergence rate of the solvers. Unlike the SPARC problem, the block Jacobi solver fails to converge, while the multigrid solver with BJ smoothing converges, demonstrating a robustness advantage of MG methods on non-axisymmetric configurations. Note, this project is under active development and these results were generated in early 2026.

Fig. 10 (left) shows the MG iteration count history for the stellarator test with $rtol = 10^{-9}$, and a solver refresh at every time step. The block Jacobi solver is used as the smoother once per iteration as the post-smoother and the pre-smoother is one iteration of PBJ, which we call a $V(0,1)$-cycle for clarity. Fig. 10 (right) shows the residual history using the failed BJ solve. This data shows that the MG solver is converging on par with the SPARC test, while the BJ solver stagnates.

5. Conclusion and future work

This work develops a toroidal semi-coarsening, geometric multigrid (MG) solver, with point-block Jacobi (PBJ) and full plane block Jacobi (BJ) smoothers, and algebraic coarse grids, in *M3D-C1*. The new MG solver is competitive on the largest version of one test case (SPARC), much faster on some smaller configurations, and is effective on a stellarator test where the BJ solver fails.

Several optimizations remain to be explored. The most urgent issue is the coarse grid data model where redundant coarse grid solves were required for the SPARC tests due to direct solver MPI failures on coarse grids during the factorization stage with both *SuperLU* and *MUMPS*. A better approach to reducing the degree of parallelism, to relieve pressure on the LU solvers, is to leave coarse grid processors idle as is often done in scalable MG solvers [24]. Additionally, redundant coarse grid solvers significantly complicate the solver parameters (Appendix B).

The most impactful optimization, and the most complex to implement, is adding $2D$ multigrid to the poloidal plane, resulting in a full $3D$ multigrid solver, which is a complex project requiring unstructured geometric coarse grids [24]. $3D$ MG will reduce the work and memory complexity of the solver substantially in coarse grid solves and the coarse grid operator construction.

Another area of improvement for this solver is expanding the options for, and size of, the blocks in the PBJ smoothers that fit in a device's thread group (e.g., an NVIDIA SM or *cooperative group*). The blocks in the current PBJ smoothers can be enlarged to some extent, so as to provide more powerful smoothers, before the cost of the dense direct solves becomes prohibitive. The appeal of PBJ smoothers is the ability to use dense linear algebra (explicit inverses and BLAS2 solves), which performs exceptionally well on GPUs. Increasing the block size adds sparsity to these sub-solves that would need to be padded, limiting the size. For example, a 3^3 vertex subdomain would have 972 equations with a small amount of padding and would be tractable on existing GPU architectures. Sparse batch solvers are a potential option for larger subdomains. PETSc does have sparse batched Krylov solvers with Jacobi preconditioners [25], and algebraic MG batched preconditioners have been added to PETSc recently.

The main choices for enlarging these blocks are 1) line subdomains (e.g., in the toroidal direction), 2) plane subdomains (e.g., in the poloidal plane), or 3) isotropic subdomains. It is possible that different subdomains would be optimal for each of the three equations (U, ω, χ) in a 3×3 block outer solver – a *FieldSplit* solver in PETSc – because each of these equations represents a distinct wave in the system and will invariably have different anisotropic characteristics. This splitting of waves also nearly decouples the three equations (Appendix C), which is required for the (3) block Gauss-Seidel iteration in a *FieldSplit* solver to converge well. More effective *FieldSplit* solvers (i.e., Schur complement based methods) are not tractable for the *M3D-C1* matrices.

Reproducibility is discussed in Appendix B.

CRediT authorship contribution statement

Mark F. Adams: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Methodology, Investigation, Conceptualization; **Jin Chen:** Writing – review & editing, Writing – original draft, Validation, Software, Resources, Investigation, Data curation; **Benjamin Sturdevant:** Writing – review & editing, Writing – original draft, Visualization, Investigation, Data curation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Mark F. Adams reports financial support was provided by E O Lawrence Berkeley National Laboratory. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Fusion Energy Sciences and Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program through the FASTMath Institute under Contract No. DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 using NERSC award ASCR-ERCAP0030112. This manuscript has been [co-]authored by Princeton University/Princeton Plasma Physics Laboratory under contract number DE-AC02-09CH11466 with the U.S. Department of Energy. The United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purpose(s). DOE SciDAC program, Center for Edge of Tokamak OPtimization (CETOP), under Award Number DE-AC02-09CH11466.

Data availability

No data was used for the research described in the article.

Appendix A. The resistive MHD equations

This appendix sketches the numerical methods in the evolution of the momentum Eq. (3) with a focus on the algebraic form of the diagonal blocks of the Jacobian matrix in *M3D-C1*, which is of the most relevance to algebraic solvers. Magnetohydrodynamics (MHD) treats a magnetized plasma as a conducting fluid with mean number density n, ion mass M_i, velocity $\mathbf{v}$ and current $\mathbf{J}$, Maxwell's equations and conservation laws (velocity moments $0-2$), of the form:

Start from the resistive MHD equations:

$$\frac{\partial n}{\partial t} + \nabla\cdot(n\mathbf{v}) = 0 \quad \text{continuity} \tag{1}$$

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla\times[\mathbf{v}\times\mathbf{B} - \eta\mathbf{J}] \quad \text{Faraday's law} \tag{2}$$

$$nM_i\frac{d\mathbf{v}}{dt} + \nabla p - \nu\nabla^2\mathbf{v} = \mathbf{J}\times\mathbf{B} \quad \text{momentum} \tag{3}$$

$$\mathbf{E} + \mathbf{v}\times\mathbf{B} = \eta\mathbf{J} + \frac{1}{ne}\left(\mathbf{J}\times\mathbf{B} - \nabla p_e - \nabla\cdot\Pi_e\right) \quad \text{Ohm's law} \tag{4}$$

$$\frac{\partial p}{\partial t} + \mathbf{v}\cdot\nabla p + \gamma p\nabla\cdot\mathbf{v} = (\gamma-1)\left[\eta\mathbf{J}^2 - \Pi : \nabla\mathbf{v}\right] \quad \text{energy in conservation form} \tag{5}$$

with electrical resistivity η, viscosity μ and a pressure tensor Π. *M3D-C1* is an extended MHD, two-fluid, model with electron charge e, an electron pressure tensor Π_e and electron pressure p_e. The total pressure p and adiabatic index γ are from the equation of state for internal energy.

A.1. Dimensionless scaling of the momentum equation

These MHD equations are scaled by physically relevant quantities, which regularizes the numerics, simplifies the notation, and results in units that are physically relevant.

$$nM_i\frac{d\mathbf{v}}{dt} + \nabla p - \nu\nabla^2\mathbf{v} = \mathbf{J}\times\mathbf{B} \tag{6}$$

$$n_0\tilde{n}M_i\frac{v_0}{t_0}\frac{d\tilde{\mathbf{v}}}{d\tilde{t}} + \frac{p_0}{l_0}\tilde{\nabla}\tilde{p} - \frac{\nu_0 v_0}{l_0^2}\tilde{\nu}\tilde{\nabla}^2\tilde{\mathbf{v}} = J_0B_0\tilde{\mathbf{J}}\times\tilde{\mathbf{B}} \tag{7}$$

$$\frac{l_0^2}{t_0^2}\frac{\mu_0 n_0 M_i}{B_0^2}\tilde{n}\frac{d\tilde{\mathbf{v}}}{d\tilde{t}} + \frac{\mu_0 p_0}{B_0^2}\tilde{\nabla}\tilde{p} - \frac{\nu_0\nu_0\mu_0}{B_0^2 l_0}\tilde{\nu}\tilde{\nabla}^2\tilde{\mathbf{v}} = \tilde{\mathbf{J}}\times\tilde{\mathbf{B}} \tag{8}$$

$$\tilde{n}\frac{d\tilde{\mathbf{v}}}{d\tilde{t}} + \tilde{\nabla}\tilde{p} - \tilde{\nu}\tilde{\nabla}^2\tilde{\mathbf{v}} = \tilde{\mathbf{J}}\times\tilde{\mathbf{B}} \tag{9}$$

where

$$t_0 = l_0\left[\frac{\mu_0 n_0 M_i}{B_0^2}\right]^{\frac{1}{2}}, p_0 = \frac{B_0^2}{\mu_0}, J_0 = \frac{B_0}{\mu_0 l_0}, \mathbf{v}_0 = \frac{l_0}{t_0}, \nu_0 = \frac{B_0^2 t_0}{\mu_0} = \frac{n_0 M_i}{t_0}l_0^2$$

A.2. Parabolization

Parabolization is critical in forming equations that are elliptic and in *M3D-C1* symmetric. This parabolization process results in a fourth order curl operator (15) and is included here for reference.

Let $\dot{\mathbf{v}} \equiv \frac{\partial \mathbf{v}}{\partial t}$ and the ideal MHD momentum equation results:

$$n_0\dot{\mathbf{v}} + \nabla p = [(\nabla\times\mathbf{B})\times\mathbf{B}] \tag{10}$$

take its time derivative

$$n_0\ddot{\mathbf{v}} + \nabla\dot{p} = \left[\left(\nabla\times\dot{\mathbf{B}}\right)\times\mathbf{B} + (\nabla\times\mathbf{B})\times\dot{\mathbf{B}}\right] \tag{11}$$

Substitute the ideal MHD equation for $\dot{\mathbf{B}}$ and $\dot{p}$

$$\dot{\mathbf{B}} = \nabla\times[\mathbf{v}\times\mathbf{B}] \tag{12}$$

$$\dot{p} = -\mathbf{v}\cdot\nabla p - \gamma p\nabla\cdot\mathbf{v} \tag{13}$$

we get the parabolization operator

$$\mathcal{L}(\mathbf{v}) = \{\nabla\times[\nabla\times(\mathbf{v}\times\mathbf{B})]\}\times\mathbf{B} + \{(\nabla\times\mathbf{B})\times\nabla\times(\mathbf{v}\times\mathbf{B})\} \tag{14}$$

$$+ \nabla(\mathbf{v}\cdot\nabla p + \gamma p\nabla\cdot\mathbf{v}) \tag{15}$$

with the parabolization, the momentum equation becomes:

$$\left\{n - \theta^2\delta t^2\mathcal{L}\right\}\frac{\partial\mathbf{v}}{\partial t} + n\mathbf{v}\cdot\nabla\mathbf{v} + \nabla p - \nu\nabla^2\mathbf{v} = [(\nabla\times\mathbf{B})\times\mathbf{B}] \tag{16}$$

Then we can advance the velocity as follows

$$\left\{n - \theta^2\delta t^2\mathcal{L}\right\}\mathbf{v}^{n+1} + \theta\delta t(n\mathbf{v}^{n+1}\cdot\nabla\mathbf{v}^n + n\mathbf{v}^n\cdot\nabla\mathbf{v}^{n+1}) - \nu\theta\delta t\nabla^2\mathbf{v}^{n+1} =$$
$$\left\{n - \theta^2\delta t^2\mathcal{L}\right\}\mathbf{v}^n - (1-2\theta)\delta t(n\mathbf{v}^n\cdot\nabla\mathbf{v}^n) + \delta t\{-\nabla p + [(\nabla\times\mathbf{B})\times\mathbf{B}]\}^{n+\frac{1}{2}} \tag{17}$$

A.3. Decompose the velocity in (R, ϕ, Z) coordinate system

An important feature of the *M3D-C1* formulation is the decomposition of velocity into three scalar fields that expand to MHD waves: U, ω, χ

$$\mathbf{v} = R^2 \nabla U \times \nabla\phi + R^2 \omega \nabla\phi + \frac{1}{R^2} \nabla_\perp \chi \tag{18}$$

The operator $\nabla_\perp$ denotes the gradient in the (R, Z) plane,

$$\nabla_\perp \chi \equiv \chi_R \hat{R} + \chi_Z \hat{Z},$$

orthogonal to $\nabla\phi$. $|\nabla\phi| = \frac{1}{R}$.

U - shear Alfven wave, does not compress the toroidal magnetic field.

ω - slow wave, does not compress the toroidal magnetic field.

χ - fast wave, does compress the toroidal field.

These equations are meant to give a sense of the nature of the equations being solved in the momentum solve.

Appendix B. Reproducibility issues, *M3D-C1* and PETSc options

M3D-C1 is open-source https://github.com/PrincetonUniversity/M3DC1, but this section is intended to aid in producing the results in this paper by *M3D-C1* users. The complete input instructions would not be feasible to include here and one would need to work with the *M3D-C1* team to set this problem up. This input deck is what needs to be provided by the solver team, the authors, and one would need to work with the physics team to reproduce these results. Details of the physical model are published by Datta [16]. The data, run and plot scripts, inputs and README-solver-paper.txt file for reproducibility data are in …mp288/jinchen/M3DC1/WORK/SPRC-m04-3D-03-TEST at NERSC.

This is an example of the PETSc input that one could provide as the `-options_file` argument to *M3D-C1*, or place in a .petscrc file. This file is for a 64 plane case. The `-pc_bjacobi_blocks 64` option must be set to the number of planes and the degree of redundancy is a tunable parameter. The block size must be set for each level (e.g., `-hard_mg_levels_1_up_redundant_pc_bjacobi_blocks 2`), and one can run with say 32 planes and the `-hard_mg_levels_6_up_pc_bjacobi_blocks 64` line will be safely ignored, thus only `-pc_bjacobi_blocks 64` need be changed for fewer planes.

```
-pc_bjacobi_blocks 64 # set to number of planes
-hard_ksp_atol 1.e-20 # (source: file)
-hard_ksp_converged_reason # (source: file)
-hard_ksp_gmres_preallocate # (source: file)
-hard_ksp_gmres_restart 200 # (source: file)
-hard_ksp_max_it 200 # (source: file)
-hard_ksp_min_it 1 # (source: command line)
-hard_ksp_monitor # (source: file)
-hard_ksp_rtol 1.e-16 # (source: command line)
-hard_ksp_type fgmres # (source: file)
-hard_ksp_view # (source: command line)
-hard_mg_coarse_ksp_type preonly # (source: file)
-hard_mg_coarse_pc_redundant_number 16 # (source: file)
-hard_mg_coarse_pc_type redundant # (source: file)
-hard_mg_coarse_redundant_ksp_converged_reason # (source: file)
-hard_mg_coarse_redundant_ksp_rtol 1.e-12 # (source: file)
-hard_mg_coarse_redundant_ksp_type gmres # (source: file)
-hard_mg_coarse_redundant_pc_factor_mat_solver_type mumps # (source: file)
-hard_mg_coarse_redundant_pc_type lu # (source: file)
-hard_mg_levels_1_up_ksp_max_it 1 # (source: file)
-hard_mg_levels_1_up_ksp_norm_type none # (source: file)
-hard_mg_levels_1_up_ksp_type gmres # (source: file)
-hard_mg_levels_1_up_pc_redundant_number 8 # (source: file)
-hard_mg_levels_1_up_pc_type redundant # (source: file)
-hard_mg_levels_1_up_redundant_ksp_type preonly # (source: file)
-hard_mg_levels_1_up_redundant_pc_bjacobi_blocks 2 # (source: file)
-hard_mg_levels_1_up_redundant_pc_type bjacobi # (source: file)
-hard_mg_levels_1_up_redundant_sub_ksp_type preonly # (source: file)
-hard_mg_levels_1_up_redundant_sub_pc_factor_mat_solver_type mumps # (source: file)
-hard_mg_levels_1_up_redundant_sub_pc_type lu # (source: file)
-hard_mg_levels_2_up_ksp_max_it 1 # (source: file)
-hard_mg_levels_2_up_ksp_norm_type none # (source: file)
-hard_mg_levels_2_up_ksp_type gmres # (source: file)
-hard_mg_levels_2_up_pc_redundant_number 4 # (source: file)
-hard_mg_levels_2_up_pc_type redundant # (source: file)
-hard_mg_levels_2_up_redundant_ksp_type preonly # (source: file)
-hard_mg_levels_2_up_redundant_pc_bjacobi_blocks 4 # (source: file)
-hard_mg_levels_2_up_redundant_pc_type bjacobi # (source: file)
-hard_mg_levels_2_up_redundant_sub_ksp_type preonly # (source: file)
-hard_mg_levels_2_up_redundant_sub_pc_factor_mat_solver_type mumps # (source: file)
-hard_mg_levels_2_up_redundant_sub_pc_type lu # (source: file)
-hard_mg_levels_3_up_ksp_max_it 1 # (source: file)
-hard_mg_levels_3_up_ksp_norm_type none # (source: file)
-hard_mg_levels_3_up_ksp_type gmres # (source: file)
-hard_mg_levels_3_up_pc_redundant_number 2 # (source: file)
-hard_mg_levels_3_up_pc_type redundant # (source: file)
-hard_mg_levels_3_up_redundant_ksp_type preonly # (source: file)
-hard_mg_levels_3_up_redundant_pc_bjacobi_blocks 8 # (source: file)
-hard_mg_levels_3_up_redundant_pc_type bjacobi # (source: file)
-hard_mg_levels_3_up_redundant_sub_ksp_type preonly # (source: file)
-hard_mg_levels_3_up_redundant_sub_pc_factor_mat_solver_type mumps # (source: file)
-hard_mg_levels_3_up_redundant_sub_pc_type lu # (source: file)
-hard_mg_levels_4_up_pc_bjacobi_blocks 16 # (source: file)
-hard_mg_levels_5_up_pc_bjacobi_blocks 32 # (source: file)
-hard_mg_levels_6_up_pc_bjacobi_blocks 64 # (source: file)
-hard_mg_levels_ksp_max_it 1 # (source: file)
-hard_mg_levels_ksp_norm_type none # (source: file)
-hard_mg_levels_ksp_type gmres # (source: file)
-hard_mg_levels_pc_type pbjacobi # (source: file)
-hard_mg_levels_up_ksp_converged_reason # (source: file)
-hard_mg_levels_up_ksp_gmres_restart 1 # (source: file)
-hard_mg_levels_up_ksp_max_it 1 # (source: file)
-hard_mg_levels_up_ksp_norm_type none # (source: file)
-hard_mg_levels_up_ksp_type gmres # (source: file)
-hard_mg_levels_up_pc_type bjacobi # (source: file)
-hard_mg_levels_up_sub_ksp_type preonly # (source: file)
-hard_mg_levels_up_sub_pc_factor_mat_solver_type mumps # (source: file)
-hard_mg_levels_up_sub_pc_type lu # (source: file)
-hard_pc_mg_distinct_smoothup # (source: file)
-ihard_mat_type aijkokkos # (source: command line)
-ksp_atol 1.e-20 # (source: file)
-ksp_converged_reason # (source: file)
-ksp_error_if_not_converged # (source: file)
-ksp_max_it 100 # (source: file)
-ksp_monitor_5 # (source: command line)
-ksp_rtol 1.e-9 # (source: file)
-ksp_type gmres # (source: file)
-log_view :log_mg16_bj.txt # (source: command line)
-mgfs 5 # (source: command line)
-mhard_mat_type aijkokkos # (source: command line)
-on_error_abort # (source: file)
-options_left # (source: file)
-pc_type bjacobi # (source: file)
-sub_ksp_type preonly # (source: file)
-sub_pc_factor_mat_solver_type mumps # (source: file)
-sub_pc_type lu # (source: file)
```

Appendix C. Block norms

This section reports the norms of these blocks, a 3×3 scalar matrix, and the degree of diagonal dominance is indicative of the convergence rates that one can expect with a *FieldSplit* solver, a block Gauss-Seidel iteration, and illustrates the near diagonalization that results from the velocity decomposition used in the momentum solver. The relative off-diagonal block norms range between 10^{-2} and 10^{-3}.

Note: these block norm tests are in the context of plane solvers. We first extracted a single block from a full 3D matrix, on which the analysis was performed.

To get a sense of the strength of coupling between fields, we consider symmetric diagonal scaling of the Frobenius norms of each field block. Specifically, we examine the following 3×3 coupling matrix for three test cases.

$$\begin{pmatrix} 1 & \frac{||A_{U,\omega}||_F}{\sqrt{||A_{U,U}||_F ||A_{\omega,\omega}||_F}} & \frac{||A_{U,\chi}||_F}{\sqrt{||A_{U,U}||_F ||A_{\chi,\chi}||_F}} \\ \frac{||A_{\omega,U}||_F}{\sqrt{||A_{U,U}||_F ||A_{\omega,\omega}||_F}} & 1 & \frac{||A_{\omega,\chi}||_F}{\sqrt{||A_{\omega,\omega}||_F ||A_{\chi,\chi}||_F}} \\ \frac{||A_{\chi,U}||_F}{\sqrt{||A_{U,U}||_F ||A_{\chi,\chi}||_F}} & \frac{||A_{\chi,\omega}||_F}{\sqrt{||A_{\omega,\omega}||_F ||A_{\chi,\chi}||_F}} & 1 \end{pmatrix}$$

The number of equations and number of non-zeros for the three test cases are given in Table C.1.

The coupling matrices for each case, RE being the model problem in this work, are:

$$\text{Case 1:} \begin{pmatrix} 1.0 & 1.5 \times 10^{-4} & 1.3 \times 10^{-3} \\ 1.5 \times 10^{-4} & 1.0 & 8.5 \times 10^{-4} \\ 1.3 \times 10^{-3} & 8.5 \times 10^{-4} & 1.0 \end{pmatrix}$$

$$\text{Case 2:} \begin{pmatrix} 1.0 & 2.3 \times 10^{-4} & 8.4 \times 10^{-3} \\ 2.3 \times 10^{-4} & 1.0 & 1.3 \times 10^{-3} \\ 8.4 \times 10^{-3} & 1.3 \times 10^{-3} & 1.0 \end{pmatrix}$$

$$\text{Case 3:} \begin{pmatrix} 1.0 & 5.1 \times 10^{-5} & 6.3 \times 10^{-3} \\ 5.1 \times 10^{-5} & 1.0 & 1.6 \times 10^{-2} \\ 6.3 \times 10^{-3} & 1.6 \times 10^{-2} & 1.0 \end{pmatrix}$$

$$\text{Case RE:} \begin{pmatrix} 1.0 & 4.6 \times 10^{-4} & 2.9 \times 10^{-2} \\ 4.6 \times 10^{-4} & 1.0 & 9.7 \times 10^{-3} \\ 2.9 \times 10^{-2} & 9.7 \times 10^{-3} & 1.0 \end{pmatrix}$$

Table C.1
Number of equations and non-zeros in three test cases from the *M3D-C1* test case library.

	Case 1	Case 2	Case 3	Case RE (model problem)
Test case name	p-cpu-16F-R01	nstx_120446	numvar.NV3.mgfs	SPARC - disruption
N	189 180	365 796	401 292	1,879,920
NNZ	46 472 313	89 557 184	98 771 994	1,409,524,416

References

[1] N.M. Ferraro, Non-Ideal Effects on the Stability and Transport of Magnetized Plasmas, Ph.d. thesis, Princeton University, Princeton, NJ, 2008. Adviser: Stephen Jardin.

[2] U. Trottenberg, C.W. Oosterlee, A. Schüller, Multigrid, Academic Press, London, 2000.

[3] M.F. Adams, R. Samtaney, A. Brandt, Toward textbook multigrid efficiency for fully implicit resistive magnetohydrodynamics, J. Comput. Phys. 229 (18) (2010). https://doi.org/10.1016/j.jcp.2010.04.024

[4] J.L. Thomas, B. Diskin, A. Brandt, Textbook multigrid efficiency for the incompressible Navier–Stokes equations: high Reynolds number wakes and boundary layers, Comput. Fluids 30 (7) (2001) 853–874.

[5] L. Chacón, An optimal, parallel, fully implicit Newton–Krylov solver for three-dimensional viscoresistive magnetohydrodynamicsa), Phys. Plasmas 15 (5) (2008) 056103. https://doi.org/10.1063/1.2838244

[6] K. Bell, A refined triangular plate bending finite element, Int. J. Numer. Methods Eng. 1 (1) (1969) 101–122. https://doi.org/10.1002/nme.1620010108

[7] S.C. Jardin, A triangular finite element with first-derivative continuity applied to fusion MHD applications, J. Comp. Phys. 200 (2004) 133–152.

[8] D.S. Harned, D.D. Schnack, Semi-implicit method for long time scale magnetohydrodynamic computations in three dimensions, J. Comput. Phys. 65 (1) (1986) 57–70. https://doi.org/10.1016/0021-9991(86)90004-5

[9] S.C. Jardin, Review of implicit methods for the magnetohydrodynamic description of magnetically confined plasmas, J. Comput. Phys. 231 (3) (2012) 822–838. Special Issue: Computational Plasma Physics, https://doi.org/10.1016/j.jcp.2010.12.025

[10] J.P.H. Goedbloed, S. Poedts, Principles of Magnetohydrodynamics: With Applications to Laboratory and Astrophysical Plasmas, Cambridge University Press, 2004.

[11] J.P. Freidberg, Ideal Magnetohydrodynamics, Plenum Press,New York, NY, 1986.

[12] M.F. Adams, Multigrid Equation Solvers for Large Scale Nonlinear Finite Element Simulations, Ph.D. dissertation, University of California, Berkeley, 1998. Tech. report UCB//CSD-99-1033.

[13] J.H. Adler, T.R. Benson, E.C. Cyr, P.E. Farrell, S.P. MacLachlan, R.S. Tuminaro, Monolithic multigrid methods for magnetohydrodynamics, SIAM J. Sci. Comput. 43 (5) (2021) S70–S91. https://doi.org/10.1137/20M1348364

[14] J.H. Adler, Y. He, X. Hu, S.P. MacLachlan, Vector-potential finite-element formulations for two-dimensional resistive magnetohydrodynamics, Comput. Math. Appl. 77 (2) (2019) 476–493. https://doi.org/10.1016/j.camwa.2018.09.051

[15] J.H. Adler, T.R. Benson, E.C. Cyr, S.P. MacLachlan, R.S. Tuminaro, Monolithic multigrid methods for two-dimensional resistive magnetohydrodynamics, SIAM J. Sci. Comput. 38 (1) (2016) B1–B24. https://doi.org/10.1137/151006135

[16] R. Datta, C. Clauser, N. Ferraro, C. Liu, R. Sweeney, R.A. Tinguely, Coupled 2D MHD and runaway electron fluid simulations of SPARC disruptions, Phys. Plasmas 32 (8) (2025) 082505. https://doi.org/10.1063/5.0272430

[17] R. Datta, C. Clauser, N. Ferraro, R. Sweeney, R.A. Tinguely, Modeling the effect of MHD activity on runaway electron generation during SPARC disruptions, 2025 arXiv:2512.05709, [physics.plasm-ph].

[18] Y. Zhou, N.M. Ferraro, S.C. Jardin, H.R. Strauss, Approach to nonlinear magnetohydrodynamic simulations in stellarator geometry, Nucl. Fusion 61 (8) (2021) 086015.

[19] S. Saxena, N. Ferraro, M.F. Martin, A.M. Wright, Bootstrap current modeling in M3D-C1, 2025. arXiv:2507.05166, [physics.plasm-ph].

[20] M. Landreman, S. Buller, M. Drevlak, Optimization of quasi-symmetric stellarators with self-consistent bootstrap current and energetic particle confinement, Phys. Plasmas 29 (8) (2022) 082501-1–082501-14.

[21] O. Sauter, C. Angioni, Y.R. Lin-Liu, Neoclassical conductivity and bootstrap current formulas for general axisymmetric equilibria and arbitrary collisionality regime, Phys. Plasmas 6 (7) (1999) 2834–2839.

[22] A. Redl, C. Angioni, E. Belli, O. Sauter, ASDEX Upgrade Team, EUROfusion MST1 Team, A new set of analytical formulae for the computation of the bootstrap current and the neoclassical conductivity in tokamaks, Phys. Plasmas 28 (2) (2021) 022502-1–022502-21.

[23] S. Saxena, N.M. Ferraro, M.F. Martin, A.M. Wright, Bootstrap current modeling in M3D-C1,(2025) arXiv preprint arXiv:2507.05166.

[24] M.F. Adams, H.H. Bayraktar, T.M. Keaveny, P. Papadopoulos, Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom, in: SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, 2004, pp. 34–34. https://doi.org/10.1109/SC.2004.62

[25] M.F. Adams, P. Wang, J. Merson, K. Huck, M.G. Knepley, A performance portable, fully implicit landau collision operator with batched linear solvers, SIAM J. Sci. Comput. 47 (2) (2025) B360–B381. https://doi.org/10.1137/24M1640252