Excalibur-Neptune report 2047356-TN-03-2

Task 1.2 Elliptic solver implementation

Ben Dudson, Peter Hill, Ed Higgins, David Dickinson, and Steven Wright

University of York

David Moxey

University of Exeter

June 17, 2021

Contents

1 Executive summary

Reference elliptic solver implementations in BOUT++ [1] are described, along with the techniques which use simplified methods to effectively precondition more complex problems. Implementations in BOUT++ include previously developed PETSc-based solvers, and a new Hypre implementation.

The results of an axisymmetric Alfvén wave test are presented, showing differences in numerical stability between direct and iterative solvers for these modes.

The Hypre-based solver has been profiled on the Lassen supercomputer [2] in collaboration with LLNL. This indicates that GPUs can result in significant speedups of the matrix solves (e.g. factor of 6.9 here, on 4 GPUs vs 40 CPUs), but that other parts of the problem setup and solve can dominate and must be

mitigated to achieve good performance.

2 Problem description

Elliptic problems appear as important elements of fluid and (gyro-)kinetic plasma models. Here we focus on the solution of the electrostatic potential ϕ, which is required in order to advance the model equations, and acts in a similar way to the stream function in incompressible fluid flow. The usual technique used to calculate ϕ in edge simulation codes is to evolve the vorticity ω in time, and at each timestep to invert a second order elliptic equation of the form:

$$\nabla \cdot \left(\frac{m_i n}{B^2} \nabla_\perp \phi \right) = \omega \tag{1}$$

where m_i is the ion mass; B is the magnetic field strength which varies in space, but typically not strongly in time; n is the plasma density, which does vary in time and space. This time variation poses a challenge for methods which require matrix assembly, because the matrix elements then change in time. In some situations and models this density is replaced by a constant. This is often called the "Boussinesq" approximation, by analogy to buoyancy-driven fluid flow.

The elliptic operator in equation 1 includes an operator $\nabla_\perp = \nabla - \mathbf{bb} \cdot \nabla$ which is the component of the gradient perpendicular to the magnetic field unit vector $\mathbf{b}$. In the case where $\mathbf{b}$ is constant, equation 1 becomes a 2D problem, but in toroidal fusion systems such as tokamaks $\mathbf{b}$ varies in space (and in principle also in time). The result is that the 2D surface becomes a complicated helix which wraps around the torus. In general this helix does not close on itself, and so the 2D surface fills the 3D domain. Equation 1 is therefore a 3D problem, but one which in some cases can be approximated by a 2D system, when gradients along the magnetic field $\mathbf{b}$ are neglected.

3 Implementation

The BOUT++ code includes several solvers for potential, because these are commonly needed in the drift-reduced fluid models it is designed to solve. The system to be inverted is in general 3-dimensional, but can be simplified in many cases. There are therefore several implementations

- 3D solvers, using PETSc and Hypre. These have been developed relatively recently; the Hypre implementation has been developed in the last year.
- 2D solvers, where the derivatives of the electrostatic potential ϕ along the magnetic field are neglected. In BOUT++ the potential is solved on toroidal planes (in ψ, ζ) rather than poloidal planes. The toroidal plane requires lower resolution, and means that the geometry coefficients are constant in one dimension (toroidal angle ζ). Disadvantages include a singularity in the coordinates near the X-point. (Note that often ϕ is often used for both toroidal angle and electrostatic potential).
- 1D solvers, where the 2D domain is Fourier decomposed into toroidal modes. This leads to a set of independent complex tridiagonal systems, one for each mode, which can be solved in parallel. This is possible in an axisymmetric system, like a tokamak, if the variation of density in toroidal angle is neglected (the Boussinesq approximation).

These approximations (neglecting parallel derivatives, and density constant in toroidal angle) tend to be good for high mode-number instabilities and turbulence, but become worse at lower mode-numbers (say toroidal mode number < 5). In particular for the axisymmetric mode these approximations become poor.

For many applications the lower dimensional applications give a good approximation, and can be trivially parallelised. They therefore make good preconditioners for the more complex problems. One quite common application of this is the "Naulin method" [3], where a fast solver using the Boussinesq approximation (constant density) is used in a Picard iteration to find the solution with varying density:

$$\nabla \cdot \left(\frac{m_i n}{B^2} \nabla_\perp \phi \right) \quad = \quad \omega$$

$$
\begin{aligned}
n &= n_0 + \delta n \\
\nabla \cdot \left(\frac{m_i n_0}{B^2} \nabla_\perp \phi^{k+1} \right) &= \omega - \nabla \cdot \left(\frac{m_i \delta n}{B^2} \nabla_\perp \phi^k \right)
\end{aligned}
$$

where n_0 is constant (in a way which simplifies the solve), and k is the iteration number. Since ϕ is evolving in time, typically this iteration starts from the solution at the previous time step, and tends to converge to acceptable tolerances in a small number of iterations (2-3).

3.1 1D solvers

If the density is assumed constant in toroidal angle, then the 2D system of equations can be Fourier decomposed into toroidal modes. Each mode is coupled in the radial (ψ) direction through a second-order ODE. Using 2nd-order central differencing, this results a complex tridiagonal system for each mode. Both direct and iterative methods are implemented in BOUT++:

Direct partitioning: This is a method described in a preprint by Austin et al [T.Austin, M.Berndt, D.Moulton, preprint LA-UR-03-4149, 2004]. For each 1D system, each processor reduces its local rows by eliminating rows until there are only two rows per processor. These rows are then gathered onto one processor, solved using the fast serial Thomas algorithm, and scattered back. The solution to this reduced system is then substituted back in to solve for the remaining rows on each processor. This results in an all-to-one and a one-to-all communication pattern. Performance is improved in the BOUT++ implementation by solving for all 1D systems in parallel, and grouping the rows into larger messages. In the current implementation the reduced systems are shared between all processors, so each processor sends the rows for some systems to one processor, some to the next processor, and so on. This balances the work, but results in all-to-all communications. On Archer2 this has been found to have much worse multi-node scaling than previous machines; This is currently being investigated with Archer2 support. A possible optimisation might be to gather onto fewer processes, which would result in more load imbalance, but fewer messages.

Cyclic Reduction: This is a direct algorithm which works like a multi-level extension of the partitioning algorithm described above. This algorithm has

very good theoretical scaling, with the number of communication steps scaling logarithmically with the number of processors. Joseph Parker (UKAEA) recently implemented this algorithm in BOUT++, using an MIT-licensed library by Ji-Hoon Kang (KAIST) [4].

Multigrid: Over the last year or so, Joseph Parker (UKAEA) has implemented two multigrid algorithms in BOUT++, including a novel variation which has been submitted for publication. These have shown significantly better performance on Archer2 than either of the direct solvers described above.

3.2 2D solvers

2D solvers do not rely on Fourier decomposition into toroidal modes, and so can take into account corrections for density or (in principle) geometry. Implementations include:

Naulin method: As described above, this method uses the 1D solver to correct for non-constant density iteratively.

Geometric multigrid: This solver was implemented under a EUROfusion HLST project. It works, but has been found to be hard to extend or diagnose due to insufficient documentation.

PETSc: The Portable, Extensible Toolkit for Scientific Computation [5] provides a wide range of linear (and nonlinear) solvers, and interfaces to third-party libraries. Code in BOUT++ constructs the matrix elements and passes the matrix to PETSc to solve. There are actually two separate 2D solvers: One which solves in the $\psi - \phi$ toroidal plane, and another (called LaplaceXY) which solves in the $\psi - \theta$ poloidal plane. The first can solve accurately for high toroidal mode numbers, while the second was added specifically in order to solve the axisymmetric component. PETSc provides many options, and while these have not been exhaustively tested, the highest performance has typically been obtained by having PETSc pass the matrix through to Hypre [6].

3.3 3D solvers

In general the equation for the potential (equation 1) is 3D, even though it contains only the component of ϕ perpendicular to the magnetic field. The axisymmetric (toroidal mode number $n = 0$) and high modes ($n > 5$) can be solved to acceptable accuracy by 2D approximations, but the low n ($1-5$) modes require a 3D solve. There are currently two solvers implemented in BOUT++:

PETSc: Chris MacMackin and John Omotani (UKAEA) implemented this in 2019, in the process creating a wrapper around PETSc to simplify the process of creating matrices and vectors from BOUT++ data structures.

Hypre: During this ExCALIBUR-Neptune project, Peter Hill has worked with staff at LLNL (particularly Steven Glenn), to adapt the PETSc interface to Hypre. This was needed in order to make use of the latest Hypre versions, which can run on GPUs. It also eliminated a small overhead in run time, and complexity of library dependencies, which came from passing data through PETSc. During this Neptune project we have worked with LLNL to characterise and optimise this solver for GPU performance on Lassen.

4 Testing

4.1 Correctness testing

The nature of the correctness tests were described in report 2047356-TN-02-2 (task 1.1). Here we report on the results of the Alfvèn wave test, comparing direct solvers ("Tri" and "LaplaceXZ", both using partitioning; see section 3.1), with an iterative solver in 2D using PETSc ("LaplaceXY", section 3.2). This was done in both shifted circle (cbm18) geometry and DIII-D X-point geometry; the X-point geometry case is the more interesting, and reported here.

Using DIII-D equilibrium, shot 119919, with grid shown in figure 1 The Alfvén wave problem (report 1.1) is simulated for the axisymmetric modes only (toroidal mode number $n = 0$).

To compare the results with and without poloidal derivatives in LaplaceXY, the

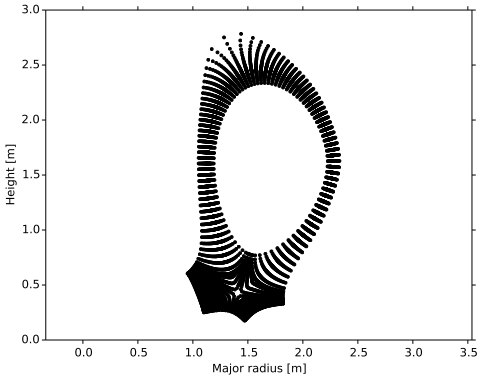

Figure 1: DIII-D 68×64 single null grid used for Alfvén wave test

simulation is run for a short time so that the vorticity is nearly the same in the two cases. The results in figure 2 compare the potential ϕ at the first radial cell outside separatrix ($x = 25$) as a function of poloidal cell index y for the Laplacian (X-Z) and LaplaceXY (X-Y) solvers: Over most of the domain the

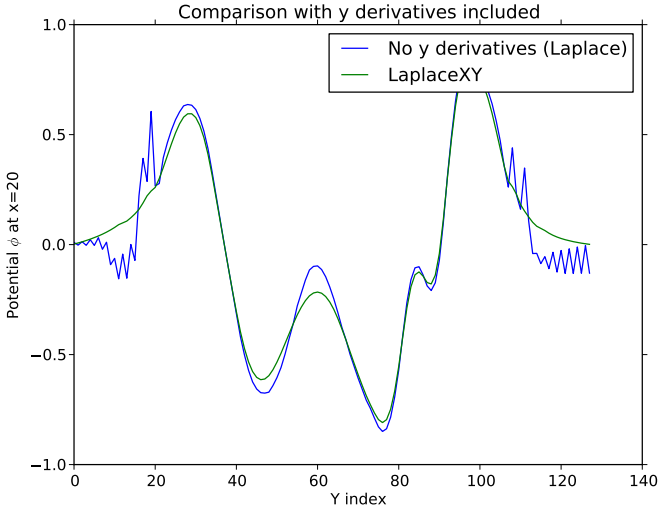

Figure 2: Electrostatic potential ϕ along the poloidal coordinate (y) just outside the separatrix. Blue line from solve neglecting poloidal derivatives; Green line from solve including poloidal derivatives.

result from the two solvers is quite close, as expected since poloidal derivatives

should be small relative to radial (x) derivative terms. The X-point branch cuts at $y = 15$ and $y = 111$ are clearly seen as locations where jumps in the potential occur when the Laplacian (X-Z) solver is used (blue line). Including the poloidal derivatives acts to smooth the potential across this location (green line).

It was found that running the simulations for longer resulted in growing instabilities in some cases but not others. The system of equations does not contain a physical growing mode, so this is the result of a numerical instability. The source of this instability would initially appear to be neglecting poloidal derivatives (blue line in figure 2) but this was not the case: Iterative solvers which neglect these derivatives were found to run stably, though with more noise than simulations which include poloidal derivatives.

Figure 3 summarises the results found: It shows a comparison between the iterative (PETSc; GMRES + SOR) `LaplaceXY` solver without poloidal derivatives; the original $X - Z$ `Laplacian` (`Tri` serial implementation) direct solver; and the `LaplaceXZ` direct solver at a point in the private flux region. Note that the

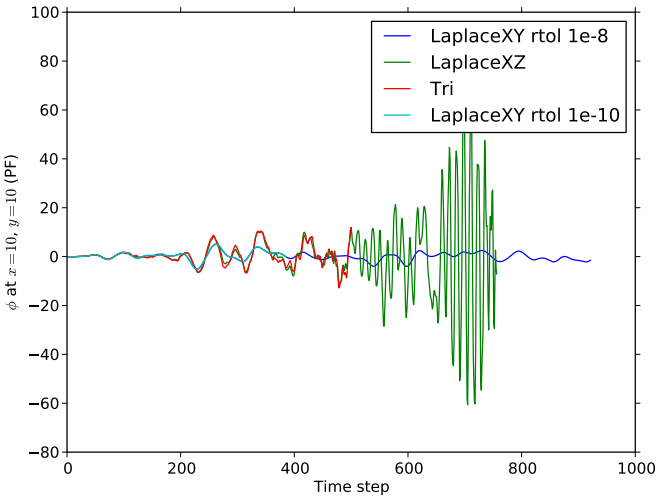

Figure 3: Comparison of solvers with poloidal derivatives removed, so only radial (x) derivatives. LaplaceXY is iterative (PETSc) while LaplaceXZ and Tri are direct; LaplaceXZ uses the same discretisation as LaplaceXY, while Tri uses a different discretisation.

direct solvers (Tri and LaplaceXZ) become unstable and have growing oscilla-

tions, while the iterative solver (LaplaceXY) remains stable. The error in the iterative method is not playing a significant role, as reducing tolerances from (rtol=1e-8, atol=1e-12) to (rtol=1e-10, atol=1e-14) makes little difference.

Conclusions drawn from these experiments are that:

- Including poloidal derivatives does not appear to be essential for the stability of $n = 0$ Alfvén wave simulations, but including them smooths the potential, and is probably important for accuracy.
- Formulating the Laplacian inversion in a conservative form (LaplaceXZ vs Tri) makes a small difference, but is probably not essential here
- There is a problem in the direct solvers for the axisymmetric component of ϕ. Direct methods tested are the serial Thomas algorithm and the direct partition algorithm. Both result in growing oscillations. This is likely because both these algorithms assume diagonal dominance, which is only marginal for axisymmetric modes when poloidal derivatives are neglected.
- The PETSc iterative solver gives very similar results to the direct solver (relative difference $\sim 10^{-6}$) on individual solves, but in the case of the direct solver this error builds up and results in a growing numerical instability.
- Preconditioning the iterative solver with a direct solver, effectively correcting the error in the direct solve, was found to remove the numerical instability.

4.2 Performance testing

Thorough performance testing of the direct and iterative tridiagonal (1D) solvers has been performed by Joseph Parker (UKAEA) on Archer 2. That work is detailed in a paper "Parallel tridiagonal matrix inversion with a hybrid multigrid–Thomas algorithm method" by J.T.Parker, P.A.Hill, D.Dickinson and B.D.Dudson, which is currently under review at the Journal of Computational and Applied Mathematics.

Performance testing of the 2D Hypre solver on Lassen has been carried out on Lassen [2] by Steven Glenn, Aaron Fisher and Holger Jones (LLNL), whom

we have worked with during this project. Lassen is a 23 PFLOP system located at Lawrence Livermoe National Laboratory (LLNL), which has an IBM Power9/NVIDIA V100 architecture similar to the Sierra and Summit supercomputers, but with 40 Power9 CPUs and 4 V100 GPUs per node.

The results in figure 4 show the timing of a Hypre solver on a large 4000x4000 mesh, on either all 40 CPUS, or all 4 GPUs of a single node. Note that in the 4 GPU case, 4 CPUs are also used, to coordinate data transfers and execute parts of the code not running on the GPU.

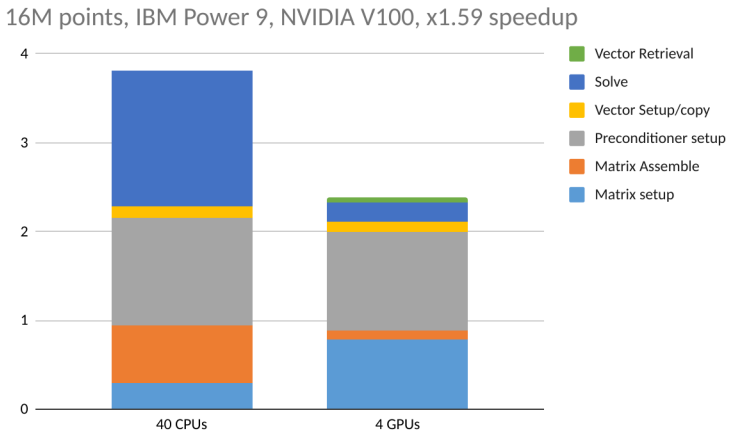

Figure 4: Timing of a Laplacian solve using Hypre on a Lasses node. Using either 40 CPUs (left) or 4 GPUs (right). Timing is broken down into stages, from matrix assembly to result vector retrieval.

Figure 4 shows that the solve time (dark blue) is 6.9 times faster on GPUs than CPUs, but is only a relatively small part of the run time (9% on GPUs; 40% on CPUs). Matrix assembly is also faster on GPUs (6x), but other parts of the solve either don't speed up (e.g preconditioner setup), or slow down (matrix setup). These steps are either not yet ported to GPUs, or require relatively complex data structure manipulation which don't tend to perform well on GPU architectures. If the matrix must be assembled every solve, this figure shows that the overall speed is only increased by a factor of 1.6.

These results indicate that matrices and preconditioners must be re-used many times, if good performance is to be achieved: Matrix and preconditioner setup takes approximately 5 times as long as vector setup, solve, and retrieval together

in this case.

As discussed in section 3, iterative methods can be used to calculate corrections, to include variations in density for example. This approach could be used to improve performance, by keeping the matrix and preconditioner on the GPU fixed, and performing a small number of iterations to correct for a time-varying matrix.

5 Conclusions

Performance and correctness tests of BOUT++ elliptic solver implementations have been described. Reference implementations using PETSc and Hypre have been shown to be robust, and can make use of GPU architectures. Direct solvers based on cyclic reduction or similar tridiagonal methods can be highly efficient, but are limited to cases where the density is assumed constant (Boussinesq approximation), and can be numerically unstable under some circumstances. These deficiencies can be corrected by employing a Picard iteration to correct for non-constant density, or by using the direct solver as a preconditioner for PETSc or Hypre's iterative methods.

6 References

[1] BOUT++ contributors. BOUT++ manual. https://bout-dev.readthedocs.io/.

[2] Livermore Computing Center. Lassen. https://hpc.llnl.gov/hardware/platforms/lassen.

[3] Magnussen, M. L. Department of Physics, Technical University of Denmark. Global numerical modeling of magnetized plasma in a linear device. https://backend.orbit.dtu.dk/ws/portalfiles/portal/133385828/, 2017.

[4] Ji-Hoon Kang. Parallel tri-diagonal matrix solver using cyclic reduction (CR), parallel CR (PCR), and Thomas+PCR hybrid algorithm. https://github.com/jihoonakang/parallel_tdma_cpp.

[5] PETSc contributors. Portable, Extensible Toolkit for Scientific Computation (PETSc). , https://www.mcs.anl.gov/petsc/.

[6] Hypre contributors. Hypre high performance preconditioners. https://github.com/hypre-space/hypre.