Unstructured finite element solvers in gyrokinetic turbulence simulations of burning plasmas

M. F. Adams¹, Y. Nishimura² and Van Emden Henson³
E-mail: adams@pppl.gov

Abstract. Many plasma physics simulation applications utilize an unstructured finite element method (FEM) discretization of an elliptic operator, of the form \(-\nabla^2 u + \alpha u = f\); when \(\alpha\) is equal to zero a “pure” Laplacian or Poisson equation results and when \(\alpha\) is greater than zero a Helmholtz equation is produced. Discretized equations of this form, often 2D, occur in many tokamak fusion plasma, or burning plasma, applications — from MHD to Gyrokinetic codes. This report investigates the performance characteristics of basic classes of linear solvers (ie, direct, one-level iterative, and multilevel iterative methods) on 2D unstructured FEM problems of the form \(-\nabla^2 u + \alpha u = f\), with both \(\alpha = 0\) and \(\alpha \neq 0\). The purpose of this work is to provide computational physicists guidelines as to appropriate linear solvers for their problems via detailed performance analysis, in terms of both the scalability and the constants in the solution times. We show, as expected, almost perfect scalability of multilevel methods and quantify the solution costs on a common computational platform — the IBM SP Power3.

1. Introduction
Tokamaks are a most promising design for burning plasma experimental reactors. The simulation of burning plasmas in tokamaks is critical to the effective design and eventual commercialization of fusion power reactors. Gyrokinetics have emerged in the past 20 years as an effective means of simulating these plasmas. Gyrokinetic formulations require a field solve for the electric static potential, as well as the Ampere’s law for the magnetic perturbation, at each time step. The field solve entails the solution of an elliptic operator of the form

\[-\nabla^2 u + \alpha u = f\]  

where \(\alpha\) is equal to zero for “pure” Poisson problems and is greater than zero for Helmholtz problems. This field solve is often discretized with an unstructured finite element method (FEM), resulting in a symmetric positive definite system of linear algebraic equations.

The earliest methods for solving the sparse linear systems of equations that arise in finite difference techniques and the finite element method, are of two types: direct methods and simple iterative techniques. Direct methods, such as Gaussian elimination, compute an exact answer (in exact arithmetic) in a finite number of operations. Iterative methods, such as Gauss-Seidel and Jacobi’s method, start with an initial guess and incrementally improve the answer until a
desired level of accuracy is achieved. Note that since one does not generally compute in exact arithmetic and that multigrid (discussed below) reduces the residual error by a constant fraction with a finite amount of work, one can thus consider multigrid to be a direct solver in floating point arithmetic. Both Gauss-Seidel and Gaussian elimination were widely used to solve these systems before electronic computers became available and were used almost exclusively well into the 1960s.

Iterative methods, such as Gauss-Seidel, iterate over the equations in the system and solve each equation for one variable with the assumption that the current solutions for the rest of the variables are correct—this is an inexpensive process with a computational complexity almost equal to that of applying the operator. A very useful class of iterative methods—Krylov subspace methods—was introduced in the 1950s [20]. The method of conjugate gradients (CG) was the first (and remains the best) Krylov method for the symmetric positive definite systems in our application. CG is inexpensive, with only a modest increase in cost per iteration over Gauss-Seidel, but usually requires preconditioning to be effective.

Preconditioning is defined as transforming the system $Ax = b$ with a non-singular matrix $M$ to $M^{-1}Ax = M^{-1}b$ or $M^{-\frac{1}{2}}AM^{-\frac{1}{2}}M^{\frac{1}{2}}x = M^{-\frac{1}{2}}b$. Jacobi preconditioning results from letting $M$ equal the diagonal of $A$. The fundamental goal in designing a preconditioner is in finding an operator $M$ such that $M^{-1}$ is inexpensive to apply and results in a matrix $M^{-1}A$ that is well conditioned and hence will converge quickly in the iterative solution process. Simple preconditioners, like Jacobi, are not scalable because as the problem size (ie, the number of unknowns $N$) increases the condition number of the preconditioned system increases in proportion to $N^{\frac{1}{2}}$ for 2D problems leading to a method with an overall computational complexity of $O(N^{3})$.

Direct methods have the advantage of being predictable, because their cost does not depend on the spectral properties (eg, condition number) of the system, but have the disadvantage that their complexity is about $O(N^{2})$ for the factorization and $O(N^{\frac{3}{2}})$ for each solve. If a system becomes extremely ill-conditioned, iterative refinement is applied, which uses the direct solver as a preconditioner for an iterative method. Thus, one can view direct methods as a class of preconditioners, where $M = A$, within an iterative framework. Of the iterative methods, Jacobi’s method uses what is essentially the cheapest non-singular preconditioner available (with the exception of the identity): $D$, the diagonal of $A$. By contrast, a direct solver uses the most expensive preconditioner available: $A$ itself. Thus, Jacobi’s method and direct solvers lie at the extreme ends of the spectrum in terms of complexity, and are both viable approaches—from this perspective it is not surprising that there are better preconditioning methods that fall between these two extremes and many such methods have been developed.

This paper investigates the most theoretically optimal class of methods: multigrid (eg, [17, 32, 7]). Multigrid methods are the most efficient iterative schemes for solving the linear systems associated with elliptic PDEs. Multigrid methods are well known to be theoretically optimal for $H^{1}$-elliptic operators, both for scalar problems like Poisson’s equation, and for systems of PDEs, such as displacement finite element discretization for elasticity [5]. Multigrid has been applied to structured grid problems for decades [6]. In the past twenty years, algebraic multigrid (AMG) methods have been developed for unstructured problems as well. See Briggs et al., and the references therein, for an introduction to multigrid methods [7]. Multigrid methods, in theory and frequently in practice, have a computational complexity of $O(N)$ making them optimal.

In this paper we investigate the performance characteristics of the primary classes of solution methods that are potentially useful for the 2D symmetric positive definite linear systems that arise in several fusion plasma simulation applications at the Princeton Plasma Physics Laboratory: 1) Jacobi’s method, 2) direct methods and 3) multigrid methods. This paper proceeds as follows: §2 introduces fusion plasma simulation applications in general and gyrokinetic methods in particular; §3 describes algebraic multigrid in general and the two methods investigated here in particular, §4 investigates performance characteristics of these
methods via numerical experiments and we conclude in §5.

2. Plasma physics simulation methods

The largest unknown in designing a tokamak reactor is the turbulence transport. Plasma particles and heat escape much faster than the time scale predicted by the classical binary collision model (so-called anomalous transport). Turbulence in plasma inherently differs from those in neutral fluids, in that it is caused by interaction of charged particles and the electromagnetic waves rather than by elastic collisions of the molecules or viscosity. Our goal is to simulate plasma turbulence in a full geometry of burning plasma experiments. Taking an International Thermonuclear Experimental Reactor (ITER) size plasma as an example, where the minor radius is on the order of one thousand ion Larmor radii, several million grid points per poloidal plane will be employed.

One of the most popular methods in simulating fully ionized plasmas is the particle in cell (PIC) method. The PIC codes evolve plasma dynamics self-consistently by alternately pushing charged particles and solving the set of Maxwell’s equations (for the electromagnetic fields). The PIC method retains important kinetic effects such as nonlinear wave-particle interactions (so-called Landau damping), which cannot be captured by the fluid models [18, 10]. Further, the invention of gyrokinetic simulation method [22] enables us to study plasma turbulence in a global scale. The basic idea behind the gyrokinetic simulation is to time-average rapid precessing motions, and only to push the guiding center motion for the particles. The finite Larmor radius effects enter the system through the gyrokinetic Poisson equation [22, 23] (GKP, hereafter). The exact form of the gyrokinetic Poisson equation is in an integral form and is solved by a linear iterative method [26] under the condition when the electrons are adiabatic [11]. Unfortunately, the iterative method cannot be applied in the presence of the non-adiabatic kinetic electrons. With non-adiabatic electrons the iteration matrix is no longer diagonally dominant. This requires a new algorithm. In its simplest form using Pade approximation [23], GKP for electrostatic perturbations can be given by a differential form

\[-\nabla^2 u = f\]  

In Eq. (2), the right side \( f \) corresponds to the fluctuating part of the charge density, \( f = \delta n_i - \delta n_e \) where \( \delta n_i \) and \( \delta n_e \) are the ion and the electron guiding center charge density, respectively. In the tokamak core regions, due to the auxiliary ion heating, the ion temperature gradient exceeds the density gradient and induces an instability known as the ITG mode which is believed to be the drive for the core turbulence. In the adiabatic electron limit (\( \delta n_e = u \)) we recover the form of Eq. (1) with \( \alpha = 1 \).

On top of the electrostatic perturbation in GKP, we need to solve Ampere’s law for the magnetic perturbation. The vector potential \( A_\parallel \) is given

\[\nabla^2 A_\parallel = -J_\parallel,\]  

where \( J_\parallel \) is the plasma current obtained from the particles. Here, the subscript denotes the direction parallel to the magnetic field.

The global gyrokinetic simulation code (GTC, [25]) employs a unique mesh structure, the so-called global field aligned mesh, which rotates together with the magnetic field line pitch. The field aligning is an important technique for global simulations (otherwise one needs to use a \( \times 100 \) finer mesh in the toroidal direction). Consequently, GTC employs a logically non-rectangular grid with a number of poloidal grid points which increases radially. At first glance the GTC grid seems like a conventional FEM grid (on each poloidal plane). However, the grid rotates in the toroidal direction with different speed at each radius and the grid structures are different at different toroidal angles (see Fig.1(b). As a consequence, the elements are distorted at different
toroidal angles. To retain the accuracy of the finite element method, the labeling of the vertices are changed depending on poloidal planes (and thus we generate multiple matrices corresponding to the number of the planes).

![Figure 1](image)

**Figure 1.** Adaptation of the FEM generator to the GTC grid. Here, very small numbers of grid points are taken to emphasize the GTC grid topology. (a) A conceptual plot of the logically non-rectangular GTC grid. Triangular finite elements with a linear shape functions are adapted. The poloidal plane is divided into quadrants (this is for the purpose of second domain decomposition). The number of the grid points increases by a constant factor of four in the radial direction. (b) A schematic diagram which suggest the field aligning. The red blocks illustrate the grid points at one toroidal angle, while the blue blocks illustrate those at the other side of the toroidal direction. Note each radius has a different rotation speed (not a rigid rotation) depending on the magnetic field line pitch.

### 3. Multigrid introduction

Multigrid is motivated first by the fact that inexpensive iterative methods, such as Gauss-Seidel, are effective at reducing high frequency, or high energy, error. These solvers are called *smoothers* because they render the error geometrically smooth by reducing its high frequency content. Smoothers are, however, ineffectual at reducing low frequency or low energy error. The second observation that motivates multigrid methods is that low energy error can be represented effectively with a coarse version of the problem and that this coarse problem can be “solved” recursively in a multigrid process. That is, the solution can be projected to a smaller space to provide a coarse grid correction, in much the same way that the finite element method computes an approximate solution by projecting the infinite dimensional solution onto a finite dimensional subspace. Multigrid is practical because this projection can be prepared and applied with reasonable and scalable cost. Like the finite element method, the design of the spaces that one uses in the projection is critical in defining a particular method.

The discrete form of these coarse grid spaces are represented in the columns of the *prolongation* operator $P$. The prolongation operator is used to map corrections, to the solution,
from the coarse grid to the fine grid. Residuals are mapped from the fine grid to the coarse grid with the restriction operator \( R \); \( R \) is generally equal to \( P^T \). The coarse grid matrix may be formed in one of two ways, either algebraically to form Galerkin (or variational) coarse grid operator \( A_{\text{coarse}} \leftarrow RA_{\text{fine}}P \) or by creating a new finite element problem on each coarse grid (if an explicit mesh is available) thereby allowing the finite element implementation to construct the matrix. The smoothers and coarse grid corrections may be applied in almost any order, either additively or multiplicatively—here we use the standard multiplicative \( V \)-cycle [31], described in Figure 2, using a smoother \( x \leftarrow S(A, b) \) along with the \( P \) and \( R \) matrices, on each level.

```plaintext
function MGV(A_i, b_i)
if there is a coarser grid \( i + 1 \)
    \( x_i \leftarrow S^\nu_1(A_i, b_i) \)  # \( \nu_1 \) iterations of the (pre) smoother
    \( r_i \leftarrow b_i - Ax_i \)  # residual calculation (not needed if \( \nu_1 = 0 \))
    \( r_{i+1} \leftarrow MGV(R_{i+1}A_iP_{i+1}, r_{i+1}) \)  # the recursive application
    \( x_i \leftarrow x_i + P_{i+1}(x_{i+1}) \)  # prolongation of coarse grid correction
    \( r_i \leftarrow b_i - A_ix_i \)
    \( x_i \leftarrow x_i + S^\nu_2(A_i, r_i) \)  # \( \nu_2 \) iterations of the (post) smoother
else
    \( x_i \leftarrow A_i^{-1}r_i \)  # direct solve of coarsest grid
return \( x_i \)
```

Figure 2. Multigrid \( V \)-cycle Algorithm

Multigrid is often used as a preconditioner for a (Krylov) iterative method; for the numerical experiments described in this paper, one multigrid \( V \)-cycle is used as the preconditioner.

For structured grid problems, the coarse grids are predefined as some geometric subset of the fine grids (e.g., dropping every second grid-point in each direction to form the coarse grid) and the operators \( R \), \( P \), and \( A_{\text{coarse}} \) are defined with simple geometric interpolation. Only two operators are required to define a multigrid method: 1) \( P \), whose columns define the coarse grid space and with \( R^T = P \), determine the coarse matrix \( A_{\text{coarse}} = RA_{\text{fine}}P \), and 2) the smoother \( S(A, b) \). When these operators can be defined automatically, using little or no knowledge of the geometry of the problem or the underlying PDE, the resulting process is known as algebraic multigrid, or AMG.

AMG methods are, by the broadest definition, methods that select the coarse grids and construct the coarse grid operators “algebraically,” usually via a Galerkin process. The coarse grid spaces and operators are typically constructed from the stiffness matrix alone, with little or no extra data required from the application. Two such methods are investigated here: 1) classical algebraic multigrid [19] (§3.1), and 2) smoothed aggregation [33] (§3.2).

### 3.1. Classical algebraic multigrid

We describe briefly the necessary ingredients and operations that make up the classical AMG algorithm. Detailed explanations may be found in [30]. Assume we are given the operator matrix, \( A \), the right-hand side \( b \), and that we have a “grid” \( \Omega \) on which the problem is posed. Note that no physical grid is required to perform AMG; at its most basic, \( \Omega \) can comprise simply a list of indices \( \{1, 2, \ldots, N\} \). Using subscripts to indicate level number, where \( 1 \) denotes the finest level (so that \( A_1 = A \) and \( \Omega_1 = \Omega \)), the components required to perform AMG are as follows:

(i) Grids \( \Omega_1 \supset \Omega_2 \supset \ldots \supset \Omega_M \).

(ii) Grid operators \( A_1, A_2, \ldots, A_M \).
(iii) Grid transfer operators:
  Prolongation $P_i$, $i = 2, 3, \ldots, M - 1$,
  Restriction $R_i$, $i = 2, 3, \ldots, M - 1$.
(iv) A smoother $S_i$, which may be defined universally or for each level, $S_i$, $i = 1, 2, \ldots, M$.

Once these components are defined, AMG follows the recursively defined multigrid V-cycle, which is given in Figure 3.1.

Since classical AMG typically uses the standard multigrid cycling, the main topic of interest here is the choice of the AMG components. This is done in a separate preprocessing step, known as the setup phase.

**AMG Setup Phase:**
(i) Set $i = 1$.
(ii) Partition $\Omega_i$ into disjoint sets $C_i$ and $F_i$.
   (a) Set $\Omega_{i+1} = C_i$.
   (b) Define interpolation $P_{i+1}$.
(iii) Set $R_{i+1} = P_{i+1}^T$ and $A_{i+1} = R_{i+1}A_iP_{i+1}$.
(iv) If $\Omega_{i+1}$ is small enough, set $M = i + 1$ and stop. Otherwise, set $i = i + 1$ and go to step 2.

3.1.1. Coarse grid selection

In the classical algorithm, coarse grid selection is the partitioning of $\Omega_i$ into the sets $C_i$ and $F_i$. Since the focus here is on coarsening a particular level, $i$, the level–subscripts are omitted. In addition to selecting the coarse points we must choose, for each fine-grid point $k \in F \equiv \Omega - C$, a small set $C(k) \subset C$ of points that will be used by $P$ to compute interpolated values at $k$.

The choice of coarse points follows from a concept of dependence. We say that the point $k$ depends on the point $j$ if, in the $k$th equation, the value of the $j$th unknown is important in determining the value of $k$th unknown. Usually this is true when the entry $a_{kj}$ of $A$ is large relative to all other off-diagonal coefficients of row $k$. We denote by $D_k$ the set of of points on which a point $k$ depends.

We try to adhere to two criteria while choosing $C$ and $F$:

**C1:** For each $k \in F$, each $j \in D_k$ is either in $C$ or $D_j \cap C(k) \neq \emptyset$.

**C2:** $C$ should be a maximal subset with the property that no point in $C$ depends on another point in $C$.

**C1** is designed to insure that the value of the $j$th unknown is represented in the interpolation formula for the $k$th unknown when $k$ depends on $j$, even if $j$ is not itself a $C$-point. **C2** is designed to control the size of the coarse grid. If the coarse grid is a large fraction of the total points in $\Omega$, then the interpolation of smooth errors is likely to be very accurate. This in turn generally produces better convergence factors. However, large coarse grids decrease the sparsities of $P, R, \text{ and } RAP$ and often entail a prohibitively large amount of work in doing $V$-cycles.

It is not always possible to enforce both **C1** and **C2**. Because the classical definition of prolongation depends on **C1**, we choose to enforce **C1** rigorously, while using **C2** as a guide. Classical coarse-grid selection thus takes two passes. The first pass begins by assigning to each point $k$ the number, $\lambda_k$, of other points that depend on $k$. We then select a point with maximal $\lambda$ as the first point in $C$. The points that depend on $k$ then become $F$-points, and since all other points that these new $F$-points depend upon are potential $C$-points, the $\lambda$ values of these other points is incremented, making them more likely to be selected as $C$-points. The process is then repeated until all points are assigned to $C$ or $F$. 
For some model problems this single pass is sufficient; however, it is not uncommon in practice that the partition achieved in this pass violates \( C_1 \), and there may be a large number of \( F-F \) dependencies between points not sharing a \( C \)-point. To alleviate this a second pass for the coarsening algorithm is done, in which each of the \( F \)-points is examined in turn; if there are \( F-F \) dependencies with points not depending on a common \( C \)-point, one of the two \( F \)-points is changed into a \( C \)-point.

3.1.2. Defining the prolongation operator Once the coarse grid \( \Omega_{i+1} \) has been partitioned into \( C_{i+1} \) and \( F_{i+1} \), it is necessary to define the prolongation operator \( P_{i+1} \) that is used to interpolate values on \( \Omega_i \). To prolong an \( \Omega_{i+1} \) vector \( x_{i+1} \) onto \( \Omega_i \), the classical AMG algorithm uses an operator of the form:

\[
(P_{i+1}x_i)_k = \begin{cases} 
(x_i)_k & \text{if } k \in C_i, \\
\sum_{j \in C(k)} w_{kj}(x_i)_j & \text{if } i \in F_i.
\end{cases}
\]

Defining the prolongation weights \( w_{kj} \) ensues from the observation that slow convergence of relaxation on a level occurs when the error is (algebraically) smooth, and this condition is equivalent to the residual, \( r_i \), being “small.” That is, \( A_i e_i \approx 0 \). Dropping the level subscript \( i \), we focus on errors satisfying

\[
a_{kk} e_k \approx -\sum_{j \neq k} a_{kj} e_j.
\]

To use this as the basis for a prolongation formula, we break this sum into two pieces: the sum over points that are in the interpolatory set \( C(k) \) (and hence are on the coarse grid), and those that are not: for \( k \in F \), the condition of “small residual” can be rewritten as:

\[
a_{kk} e_k \approx -\sum_{m \in C(k)} a_{km} e_m - \sum_{m \notin C(k)} a_{km} e_m.
\]

AMG prolongation is defined by making the following approximations in the last term in (5):

\[
\forall m \notin C(k), \quad e_m \approx \begin{cases} 
e_k & \text{if } m \in D_k \\
\sum_{j \in C(k)} a_{kj} e_j / \sum_{j \in C(k)} a_{kj} & \text{otherwise}.
\end{cases}
\]

Substituting this into (5) and solving for \( e_k \) gives the desired interpolation weights for point \( k \in F \).

3.1.3. Assessing the algorithm Determining the quality of an AMG algorithm is not an easy task, because of the complicated interplay of the various components. Three important measures are used in judging the quality of an AMG algorithm. One obvious measure is the asymptotic convergence factor (per V-cycle) that results. A second is grid complexity, defined as the ratio of the total number of gridpoints, on all grid levels, to the number of gridpoints on the original fine grid. Grid complexity is a measure of the storage space needed in the algorithm. The third measure is operator complexity, defined as the ratio of total number of nonzero entries in the operator matrices \( A_i \), for all levels \( i \), to the number of nonzeros in the fine-grid operator \( A = A_1 \). Operator complexity also gives a measure of the storage cost, and in addition, since the most expensive work in the cycle is relaxation, which is proportional to the number of nonzero
entries in the operator matrices, operator complexity measures the cost of the V-cycle in terms of operation count. All three of these factors are also influenced heavily by various parts of the AMG algorithm. Coarse-grid selection and the prolongation operator are interdependent, in that the number of coarse points affects the number of nonzeros in $P_{i+1}$ and in the coarse operator $A_{i+1}$. Hence the cost of prolongation and both types of complexity are affected. The quality of prolongation obviously affects the convergence rate, and the selection of the coarse points affects the ability for $P_{i+1}$ to accurately transfer corrections to the fine grid.

The classical coarsening algorithm tends to produce very good coarsening when applied to standard geometric problems [30, 12]. In particular, it has the property that it tends to coarsen in the direction of dependence even in cases of anisotropic operators and discontinuous coefficients [8]. In the standard model problems it produces the same coarse grids, operator and grid complexities, and convergence factors that are typically seen in geometric methods [8].

### 3.2. Smoothed aggregation multigrid

Aggregation multigrid methods require the null space or kernel of the operator (without Dirichlet boundary conditions applied). For the Poisson operators consider here this is simply the constant vector (eg, a vector of all ones). Many plane aggregation methods have been developed [9, 14, 15]. Aggregation methods, as the name implies, aggregate nodes and then inject the kernel vectors onto these nodes sets resulting in piecewise constant coarse grid space functions. These aggregates are constructed so that the nodes within an aggregate are “strongly connected” (see §3.2.1 and [33]).

An important improvement in plane aggregation methods is the addition of smoothing which provides significant improvement to the convergence bounds of these methods [33, 27], and is especially effective in practice on more challenging problems [1].

The smoothed aggregation algorithm proceeds as follows: starting on fine grid $i = 1$ with provided kernel vectors in the matrix $B_1$ (an $n$ by 1 matrix with the constant vector):

(i) Construct aggregates (a nodal partitioning) on the current (fine) grid $i$, as described in §3.2.1

(ii) For each aggregate $J$ extract the submatrix $B_i^J$ of $B_i$ associated with the nodes in aggregate $J$

(iii) On each aggregate $J$ construct the initial prolongator $\bar{P}_i^J$ with a QR factorization: $B_i^J \rightarrow \bar{P}_i^J B_i^J$

(iv) $B_i^J$ is a 1 by 1 matrix for scalar operators, and is used as the rows of the kernel matrix associated with the coarse grid node $J$ on the next grid $i+1$ — this provides the mechanism to apply the method recursively

(v) The initial prolongator $\bar{P}_i$ for grid $i$ is a (tall skinny) block diagonal matrix and is formed by “injecting” $\bar{P}_i^J$ into the columns associated with node $J$, that is $\bar{P}_i^J$ is the $J^{th}$ diagonal block of $\bar{P}_i$

(vi) Each column of $\bar{P}_i$ is “smoothed” with one iteration of an iterative method to provide the prolongator $P_i$ for grid $i$: $P_i \leftarrow (I - \tau D_i^{-1} A_i) \bar{P}_i$

(vii) The next grid operator is constructed algebraically: $A_{i+1} \leftarrow P_i^T A_i P_i$

This algorithm provides all of the operators: $P_i$, $R_i = P_i^T$, and $A_i$. $D_i$ above can be any symmetric positive definite preconditioner matrix. The nodal block diagonal of $A_i$ (ie, 1 by 1 diagonal blocks of $A$) is used in this paper and $\tau = \frac{\lambda_i^2}{\lambda_i^5}$ where $\lambda_i$ is an estimate of the highest eigenvalue of $D_i^{-1} A_i$ (see [27] for details). The construction of the aggregates in step 1 above is the last item in the algorithm that remains to be specified.
3.2.1. Construction of aggregates

The construction of the aggregates in smoothed aggregation is an important aspect of the implementation because the choice of aggregates can significantly affect the convergence rate and the memory complexity of the coarse grids which significantly effects the complexity of each iteration and the setup cost (ie, coarse grid construction). We employ a maximal independent set (MIS) with post-processing similar to the original algorithm proposed by Vanek et al.[33]. Vanek recommends that aggregates should be “connected by a path of strong coupling” as a means of automatically achieving semi-coarsening (semi-coarsening is an effective method for anisotropic problems such as some fluid flow problems [28, 13, 16]). Our experience indicates that the selection of strongly connected aggregates is effective for solid mechanics problems with large jumps in material coefficients as well.

Our algorithm proceeds as follows: given a norm $\|\cdot\|$, edge weight $w_{ij}$ between two nodes $i$ and $j$ are computed with $w_{ij} = \frac{\|a_{ij}\|}{\sqrt{\|a_{ii}\|\|a_{jj}\|}}$, where $a_{ij}$ is the $d \times d$ sub-matrix of the stiffness matrix that is associated with the degrees of freedom of nodes $i$ and node $j$ (here $d$ is the number of degrees of freedom per node on the grid, ie, 1 for our test problems). Note that for symmetric positive definite problems $w_{ij} \leq 1$ if the two–norm $\|a_{ij}\|_2$ is used; here for simplicity we use an average of the one and infinity norms $\|a_{ij}\| = \frac{\|a_{ij}\|_1 + \|a_{ij}\|_\infty}{2}$. The MIS is computed with a graph that has been modified by dropping edges that have weights that fall below a certain threshold $\epsilon$; we use $\epsilon = 0.08 \cdot \left(\frac{1}{2}\right)^{l-1}$, where $l$ is the grid number as suggested by Vanek [33].

Common “greedy” MIS algorithms naturally construct a nodal partitioning; these partitions, however, tend to be too large on 3D problems (ie, the aggregates are too small) for smoothed aggregation because the complexity of the coarse grids tend to be larger than that which is optimal for the overall complexity of the solver. A post-processing step is thus advisable to increase the size of the aggregates. We iterate over the aggregates and coalesce the nodes in the smallest aggregates with nearby aggregates with which each node has the largest sum of edge weights, constrained by requiring that the resulting aggregate be less than two times the minimum degree of nodes in the original aggregate. This heuristic is used to limit the size of aggregates as large aggregates would create a “bottleneck” in the convergence of the solver as the residual would be relatively poorly resolved by the coarse grid correction on these large aggregates. The minimum degree term is meant to reflect the lower rate of coarsening that the MIS provides in 2D problems and that is desirable for the convergence rate of the solver.

4. Numerical results

Here we briefly discuss the background of the physics problem and the numerical simulation before demonstrating the effectiveness of the algebraic multigrid preconditioners. The Gyrokinetic Toroidal Code (GTC) we employed is designed to study short wave length micro-instabilities. The code is developed in Fortran90. GTC is one of the most advanced gyrokinetic simulation codes, which can cover the entire tokamak geometry with a practical velocity space resolution. Parameters for the ion temperature gradient (ITG) mode are used in this study, where we assume electrons to be adiabatic ($\delta n_e = u$).

After scattering Monte-Carlo particle samples into the configuration space and the velocity space at $t = 0$ (by a random number generator), the particle simulation method advance as follows: (1) particles are pushed by solving the guiding center equations, (2) particle charges are gathered on the grid points, (3) the gyrokinetic Poisson equation is solved to obtain $u$, (4) the electric field calculated through the relation $E = -\nabla u$, and the $E$ values are used for the push in the first step. The processes (1) to (4) is repeated to evolve the system self-consistently, for up to tens of thousands of time steps (for typical production runs). In this work we solve the first 20 steps. Since we employ the random number generator at $t = 0$, the particle distribution throughout these 20 steps has a similar profile to that of the turbulence state. Note in the particle simulations, the charge densities $f$ change with time and thus the right side of Eq. (1)
needed to be calculated inside the particle code each time, while the geometrical information contained in $A$ stays the same throughout the simulation.

As a reference, the typical relevant parameters used for the plasma are: toroidal magnetic field $1.91T$, equilibrium ion and electron temperature $T_i = T_e = 2500eV$, equilibrium ion and electron density $4.6 \times 10^{19} m^{-3}$. As in realistic tokamak discharges, the safety factor varies within the range of $1.53 \leq q \leq 3.58$. The device size is given by a major radius of $0.93m$ and a minor radius of $0.33m$. Annular simulating domains are taken $0.165 \leq r \leq 0.33m$. The geometry we employ is toroidal, with a circular cross section. For different cases (38K, 120K, and 640K) both grid resolution in the radial and the poloidal directions are doubled (thus the number of grid points $N$ per plane increases by a factor of four). For the $N=1.2M$ case, $0.0825 \leq r \leq 0.4125$ is taken. $N$ corresponds to the number of equations in the subsequent discussion. Figure 3 shows a typical contour plot for the electrostatic potential from a GTC simulation.

![Figure 3. A typical contour plot for the electrostatic potential from a GTC simulation](image)

Four methods of solution are investigated, a direct solver and CG with three different preconditioners: 1) diagonal (Jacobi), 2) classical AMG, and 3) smoothed aggregation AMG. The direct solver is the SuperLU package from Lawrence Berkeley National Laboratory [24]. The classical algebraic multigrid method is implemented in the HYPRE solver package from Lawrence Livermore National Laboratory [19]. The smoothed aggregation preconditioner is implemented in the package Prometheus from Columbia University [29]. Prometheus is built on the parallel numerical package PETSc [4], and the mesh partitioner Parmetis [21]. The Jacobi method (diagonal preconditioning) is a built-in preconditioner in PETSc, and all other solvers have a PETSc interface that allows them to be used as a PETSc PC object. These tests are run on the IBM SP Power3 (Seaborg) at NERSC.
4.1. Multigrid Smoothers
An important aspect of the performance of a multigrid solver, after the coarse grid spaces have been selected, is the smoother algorithm and implementation. A few words are in order to understand the details of the multigrid solvers used in this study. Gauss-Seidel is an example of what is called a multiplicative method while Jacobi is an additive method. Multiplicative methods are ideal as multigrid smoothers; additive methods require damping to be theoretically valid [1]. The matrices investigated here are M-matrices (linear finite element discretization of Poisson’s equation) and in practice, because of details of their spectral properties, they are not very sensitive to block additive smoothers if the blocks are of sufficient size, which they are in test problems used in this study. The smoothers (discussed below), while not identical, for each AMG method, provide similar smoothing properties. Hence, the convergence rates reported here accurately reflect the performance of the two AMG methods.

The default HYPRE smoother is a processor block additive Schwarz (or block Jacobi) method with one application of symmetric Gauss-Seidel as the (processor) subdomain solver. This amounts to two iterations of Gauss-Seidel as the pre and post smoother locally on each processor. The default smoother in Prometheus is a Jacobi preconditioned Chebyshev smoother—the Chebyshev polynomial essentially provides the damping that is required of an additive method when used as a multigrid smoother [3]. A second order Chebyshev polynomial is used for both pre and post smoothing in Prometheus. Thus, the computational complexity of the HYPRE and Prometheus smoothers are about equal and the iteration counts are a good relative measure of the mathematical effectiveness of the respective AMG methods.

4.2. Scaled speedup study
Scalability is investigated with a scaled speedup study using several versions of a GTC cross section. Scaled speedup, or week speedup, measures performance for several discretizations of a particular problem using a number of processors for each case to keep approximately the same number of equation per processor. This study uses a number of processors to keep about 38,000 (38K) equations per processor. To conserve computer resources, these problems were run with 20 times steps, which is about a factor of 100 times fewer that would be run in practice, and with only 8 poloidal planes instead of the normal 64. For each time step, one system of the type under investigation (i.e., Helmholtz or Poisson) is solved to a relative residual tolerance of 10^{-6}, that is, convergence is declared when the two norm of the residual \( r (r \equiv b - A\hat{x}) \) is less than 10^{-6} \( \| b \|_2 \). Note that the direct solvers provide a much smaller tolerance, about 10^{-14} \( \| b \|_2 \), due to their exact nature. The number of time steps is large enough to amortize the setup phase in the preconditioners which is trivial for the Jacobi preconditioner, significant but scalable for the two multigrid methods and about \( O(N^2) \) for the direct solver. These setup costs are not investigated carefully because they are not important in this application due to the larger number of time steps in a typical GTC run.

4.3. Pure Poisson problem
This section investigates to performance of linear solvers on the pure Poisson problem, that is with \( \alpha = 0 \) in Eq. (1). Figure 4 (left) shows the solve times for four cases, ranging in size from 38K equation to 1.2M equations. Figure 4 (right) show the iteration count vs. number of processors. This data shows that the iteration count for the direct solver is always one and the iteration counts for the two multigrid methods are roughly constant. This is to be expected because multigrid methods are optimal in that the condition number of the preconditioned system in independent of the scale of the problem. The solve times are going up slightly for the two multigrid methods which show that they are not demonstrating any significant parallel inefficiency (on Seaborg) for this problem. The solve times for the direct solver increase, as is expected, with a complexity of about \( O(N^3) \). The iteration counts, and hence the solve times,
Poisson Solve Time (20 time steps)  
Number of equations (x1000)  

Poisson Solve Times  
Jacobi  
SuperLU  
HYPRE  
Prometheus

Poisson Iterations  
Number of equations (x1000)  

Poisson Iterations  
Jacobi  
SuperLU  
HYPRE  
Prometheus

Figure 4. Pure Poisson ($\alpha = 0$)

for the Jacobi method should, according to theory, go up $O(N^{1.5})$. We do indeed see growth of this approximate order.

4.4. Helmholtz problem

This section investigates to performance of methods on the Helmholtz problem, that is with $\alpha = 1.0$ in Eq. (1). Figure 5 (left) shows the solve times for four cases going from 38K equation to 1.2M equations, and figure 4 (right) show the iteration count vs. number of processors. This

Helmholtz Solve Time (20 time steps)  
Number of equations (x1000)  

Helmholtz Solve Times  
Jacobi  
SuperLU  
HYPRE  
Prometheus

Helmholtz Iterations  
Number of equations (x1000)  

Helmholtz Iterations  
Jacobi  
SuperLU  
HYPRE  
Prometheus

Figure 5. Helmholtz ($\alpha = 1.0$)

data shows that, as expected the iteration counts for all solution methods is roughly constant, and though the Jacobi method is rising there is some evidence of it asymptotically approaching a constant. This is as expected because these problems are spectrally equivalent to the identity and thus simple preconditioners work well. Once again we observe the solve times rising only slightly for the two multigrid methods, showing that they do not suffer significant parallel inefficiency.
Again, the solve times for the direct solver are increasing as is expected with a complexity of about $O(N^2)$. 

4.5. Total times
This section investigates to total times for all parts of the time evolution in GTC using the pure Poisson field solve. This data does not include the initial setup times for the problem because these times are amortized to the point of insignificance in a typical GTC run with tens of thousands of time steps. These GTC runs used four particles per grid point or about 38 million particles for the largest problem. Five computational phases are measured:

(i) **Smooth**: High frequency filter (not parallelized).
(ii) **Field**: Compute the gradient of the potential for the electric field (not parallelized).
(iii) **Charge**: Deposit charge onto the finite element mesh.
(iv) **Solve**: The linear solve for the potential - the interest of this paper.
(v) **Push**: Push the partials with the electric field.

The Field and Smooth phases are not parallelized within each poloidal plane and because the number of processors increases by a factor of 32 one would expect the times for these two sections to increase by a factor of about 32. The Charge phase has some non-parallelized work which results in poor scalability.

Figure 6 shows the solve times for the pure Poisson case going from 38K equation to 1.2M equations. This data show that the Charge phase is not scaling well, with a parallel efficiency of about 12%.

The Smooth phase exhibits erratic times. Of the eight runs used in this study, the fastest time in the Smooth phase was 3.7 seconds — this is a factor of six times smaller than the time in the run used in Figure 6. The IBMs do exhibit somewhat nondeterministic performance and there could be some load imbalance in other parts of the code that accumulated in the synchronization.

![Figure 6. Total times](image)
points in the Smooth phase, but this data is not well understood. The Smooth and Field phases, which are not parallelized within each poloidal plane, are scaling about as expected (i.e., about 3% parallel efficiency, if the fastest measured Smooth phase time on the smallest problem is used as the base case). The push and solve phase are scaling with a parallel efficiency of about 50% from one to 32 processors.

The solve phase inefficiencies are due to several factors. First, we do observe a slight rise in iteration count. Also, there is some all-to-all communication (to communicate the solution to all processors in a plane) outside of the linear solver – this is required because the entire GTC process is not parallelized and each processor requires the entire solution. The number of equations per processor is very small (about 38K) – this requires less than 3 Mb of storage for the stiffness matrix on a machine with 1Gb of memory per processor. In our experience multigrid solver can achieve over 90% efficiency on the IBM SP Power 3 machines [2] when a significant amount of memory per processor is used.

5. Conclusion
We have solved gyrokinetic turbulence simulations of burning plasmas with up to 1.2M grid points per plane, which is enough to resolve the ion Larmor radius scale in the core of the ITER size tokamak. Scalability has been demonstrated with a wide range of processors number (from one to 32) with very few equations per processor (about 38,000). Applications to shaped plasma (PPPL’s NSTX for example) is straightforward with the unstructured grid finite element method approach. Partitioning of the matrix (corresponds to a 2nd domain decomposition in the poloidal plane) is done for the linear potential solve phase but remains to be done for the Smooth and Field phases.

Acknowledgments. The authors would like to thank Dr. Wei-Li Lee, Dr. Zhihong Lin, Dr. Stephane Ethier for their work in developing the GTC application. This work was partially funded through the Departement of Energy under contract FG02-04ER25652.

References to printed journal articles
[1] M. F. Adams, Evaluation of three unstructured multigrid methods on 3D finite element problems in solid mechanics, International Journal for Numerical Methods in Engineering, 55 (2002), pp. 519–534.
[2] M. F. Adams, H. Bayraktar, T. Keaveny, and P. Papadopoulos, Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom, in ACM/IEEE Proceedings of SC2004: High Performance Networking and Computing, 2004. Gordon Bell prize winner, special category.
[3] M. F. Adams, M. Brezina, J. J. Hu, and R. S. Tuminaro, Parallel multigrid smoothing: polynomial versus Gauss–Seidel, J. Comp. Phys., 188 (2003), pp. 593–610.
[4] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, PETSc 2.0 users manual, tech. rep., Argonne National Laboratory, 1996.
[5] D. Braess, On the combination of the multigrid method and conjugate gradients, in Multigrid Methods II, W. Hackbusch and U. Trottenberg, eds., Berlin, 1986, Springer–Verlag, pp. 52–64.
[6] A. Brandt, Multi-level adaptive solutions to boundary value problems, Math. Comput., 31 (1977), pp. 333–390.
[7] W. L. Briggs, V. E. Henson, and S. McCormick, A multigrid tutorial, Second Edition, SIAM, Philadelphia, 2000.
[8] W. L. Briggs, V. E. Henson, and S. F. McCormick, Multigrid Tutorial, SIAM, Philadelphia, PA, 2 ed., 2000.
[9] V. E. Bulgakov and G. Kuhn, High-performance multilevel iterative aggregation solver for large finite-element structural analysis problems, International Journal for Numerical Methods in Engineering, 38 (1995), pp. 3529–3544.
[10] Z. Chang and J. D. Callen, Unified fluid kinetic description of plasma microinstabilities. 1. basic equations in a sheared slab geometry, Phys. Fluids B, 5 (1992), pp. 1167–1181.
[11] F. F. Chen, Introduction to Plasma Physics and Controlled Fusion, Plenum Press, New York, 1983. p. 218. Here, adiabatic means electron temperature obeys the Boltzmann relation very closely and the phase difference between the density and the potential fluctuation vanishes.
A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, G. N. Miranda, and J. W. Ruge, Robustness and scalability of algebraic multigrid, SIAM Journal on Scientific Computing, (2000), pp. 1886–1908.

J. Dendy, M. Ida, and J. Rutledge, A semicoarsening multigrid algorithm for SIMD machines, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 1460–1469.

C. Farhat and P.-X. Roux, A method of finite element tearing and interconnecting and its parallel solution algorithm, International Journal for Numerical Methods in Engineering, 32 (1991), pp. 1205–1227.

J. Fish and V. Belsky, Generalized aggregation multilevel solver, International Journal for Numerical Methods in Engineering, 40 (1997), pp. 4341–4361.

W. Hackbusch, Multi-grid Methods and Applications, Springer–Verlag, Berlin, 1985.

G. W. Hammett and F. W. Perkins, Fluid moment models for landau damping with application to the ion-temperature-gradient instability, Phys. Rev. Lett., 66 (1990), pp. 3019–3022.

V. Henson and U. Yang, BoomerAMG: A parallel algebraic multigrid solver and preconditioner, Tech. Rep. UCRL-JC-141495, Lawrence Livermore National Laboratory, 2000.

M. R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Natl. Bur. Stand., 49 (1954), pp. 409–436.

G. Karypis and V. Kumar, Parallel multilevel k-way partitioning scheme for irregular graphs, ACM/IEEE Proceedings of SC1996: High Performance Networking and Computing, (1996).

W. W. Lee, Gyrokinetic approach in particle simulation, Phys. Fluids, (1983), pp. 556–562.

X. S. Li and J. W. Demmel, SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems, ACM Trans. Mathematical Software, 29 (2003), pp. 110–140.

Z. Lin, T. S. Haeh, W. W. Lee, W. M. Tang, and R. B. White, Turbulent transport reduction by zonal flows: Massively parallel simulations, Science, 281 (1998), pp. 1835–1837.

Z. Lin and W. W. Lee, Method for solving the gyrokinetic poisson equation in general geometry, Phys. Rev. E, 52 (1995), pp. 5646–5652.

J. Mandel, M. Brezina, and P. Vaněk, Energy optimization of algebraic multigrid bases, Tech. Rep. 125, UCD/CCM, February 1998.

D. Mavriplis, Directional agglomeration multigrid techniques for high-Reynolds number viscous flows, AIAA Journal, 37 (1999), pp. 1222–1230.

Prometheus. www.columbia.edu/~ma2325.

J. W. Ruge and K. Stüben, Algebraic multigrid (AMG), in Frontiers in Applied Mathematics, S. F. McCormick, ed., vol. 3, SIAM, Philadelphia, PA, 1987, pp. 73–130.

B. Smith, P. Bjorstad, and W. Gropp, Domain Decomposition, Cambridge University Press, 1996.

U. Trottenberg, C. Oosterlee, and A. Schüller, Multigrid, Academic Press, London, 2001.

P. Vaněk, J. Mandel, and M. Brezina, Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems, in 7th Copper Mountain Conference on Multigrid Methods, 1995.