COMMUNICATIONS IN COMPUTATIONAL PHYSICS
Vol. **2**, No. 5, pp. 881-899

Commun. Comput. Phys.
October 2007

Parallel Algebraic Multigrid Methods in Gyrokinetic Turbulence Simulations

M. F. Adams[1] and Y. Nishimura[2,*]

[1] *Columbia University, APAM, 500 W. 120th St. Rm 200, MC 4701, New York, NY 10027, USA.*
[2] *Department of Physics and Astronomy, University of California, Irvine, CA 92697-4575, USA.*

Received 13 September 2006; Accepted (in revised version) 20 January 2007

Communicated by Zhihong Lin

Available online 20 March 2007

Abstract. Parallel algebraic multigrid methods in gyrokinetic turbulence simulations are presented. Discretized equations of the elliptic operator $-\nabla^2 u + \alpha u = f$ (with both $\alpha = 0$ and $\alpha \neq 0$) are ubiquitous in magnetically confined fusion plasma applications. When α is equal to zero a "pure" Laplacian or Poisson equation results and when α is greater than zero a so called Helmholtz equation is produced. Taking a gyrokinetic turbulence simulation model as a testbed, we investigate the performance characteristics of basic classes of linear solvers (direct, one-level iterative, and multilevel iterative methods) on 2D unstructured finite element method (FEM) problems for both the Poisson and the Helmholtz equations.

PACS (2006): 52.30.Gz, 52.35.Ra, 52.65.Rr

Key words: Algebraic multigrid, gyrokinetic Poisson equation, particle in cell simulation.

1 Introduction

The largest unknown in designing a tokamak reactor is the turbulent plasma transport. Plasma particles and heat escape much faster than the time scale predicted by the classical binary collision model. Turbulence in plasma inherently differs from those in neutral fluids, in that the interaction of charged particles and the electromagnetic waves plays an important role as the instability drive as well as the turbulence regulation (or the dissipation) mechanism.

*Corresponding author. *Email addresses:* `adams@pppl.gov` (M. F. Adams), `nishimuy@uci.edu` (Y. Nishimura)

One of the popular methods in simulating fully ionized plasmas is the particle in cell (PIC) method [15]. The PIC codes evolve plasma dynamics self-consistently by alternately *pushing* charged particles and *solving the fields* that are the set of Maxwell's equations for the electromagnetic fields. The PIC method retains important kinetic effects such as nonlinear wave-particle interactions (nonlinear Landau damping), which cannot be captured by the fluid models [12,23]. While traditionally PIC methods were restricted to local phenomena, the invention of the gyrokinetic simulation method [29] enabled us to study plasma turbulence in global scales. The basic idea behind the gyrokinetic simulation method is to time-average rapid precessing motions, and only to push the guiding center motion for the particles. Instead, the finite Larmor radius effects enter the system through the *gyrokinetic Poisson equation* [29,30]. The gyrokinetic Poisson equation is in an integral form and is solved with a linear iterative method [34], under the condition when the electrons are adiabatic [16]. Unfortunately, this iterative method fails in the presence of electromagnetic effects or shear Alfven dynamics [6]. In the latter case the iteration matrix is no longer diagonally dominant and a new linear solution method is required.

The particle in cell codes keep the numbers of particles per cell nearly constant [15]. Consequently global PIC code (GTC for example) in toroidal geometry employs a logically non-rectangular grid with a number of poloidal grid points increasing in the radial direction. In this work, a finite element method (FEM) is employed for the elliptic solver [37]. In general, the FEM is suitable for dealing with complicated geometries, where unstructured meshes are employed. For example, for the International Thermonuclear Experimental Reactor (ITER) size plasma [27], where the minor radius is on the order of one thousand ion Larmor radii, several million grid points per poloidal plane are needed. For the practical application for the ITER size burning plasmas the time taken by the field solver becomes significant if naive solvers are employed. Thus, time taken for the field solver becomes an issue in plasma simulations (fluid or kinetic) with larger the grid number [42]. Successful preconditioning of the linear solvers is a cornerstone of the gyrokinetic turbulence simulations. In this paper we introduce algebraic multigrid (AMG) methods and demonstrate its efficiency for the plasma physics application. We would like to note that the FEM solver and AMG developed in this work is also applicable for a general particle in cell code [15] as well as Vlasov type simulation method [13].

The field solve entails the solution of an elliptic operator of the form

$$-\nabla^2 u + \alpha u = f, \tag{1.1}$$

where α is equal to zero for "pure" Poisson problems and is greater than zero for the so called Helmholtz problems. As we discuss below, in our gyrokinetic simulation, u represents the field quantities such as Φ (electrostatic potential) and $A_\parallel$ (vector potential), while f represents the source terms such as the charge density and the current density. This field solve is often discretized with an unstructured 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. Additionally, some Krylov iterative methods (discussed below) satisfy this definition of an exact solver but are very sensitive to roundoff error and are not practical as direct methods [26]. 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.

A useful class of iterative methods—Krylov subspace methods—was introduced in the 1950s [25]. 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 a symmetric positive definite matrix $M^{-1/2}AM^{-1/2}M^{1/2}x=M^{-1/2}b$. 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 or more precisely will converge quickly in the iterative solution process. Simple preconditioners, like Jacobi, are not scalable because as the problem size (i.e., the number of unknowns N) increases the condition number of the preconditioned system increases in proportion to $N^{1/2}$ for 2D problems leading to a method with an overall computational complexity of $\mathcal{O}(N^{3/2})$. Most simple preconditioners are, however, memory scalable and scalable in the parallel computing sense in that they can be easily parallelized and achieve high parallel efficiencies. Direct methods have the advantage of being predictable, because their cost does not depend on the spectral properties (e.g., condition number) of the system, but have the disadvantage that they are not scalable in that their complexity is about $\mathcal{O}(N^2)$ for the factorization and $\mathcal{O}(N^{3/2})$ for each solve, depending on the individual problem topology and node ordering method.

This paper investigates the most theoretically optimal class of methods: multigrid (e.g., [10, 21, 43]). 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 [8]. Multigrid is also efficient in terms of memory, especially on 3D problems, when compared to direct methods. Multigrid has been applied to structured grid problems for decades [9]. 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 [10]. Multigrid methods, in theory and frequently in practice, have a computational complexity of $\mathcal{O}(N)$ 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 gyrokinetic simulations by: 1) CG preconditioned by Jacobi's method, 2) direct methods and 3) CG preconditioned by multigrid methods. This paper proceeds as follows: Section 2 introduces gyrokinetic simulation methods; Section 3 describes algebraic multigrid in general and the two methods investigated here in particular; Section 4 investigates performance characteristics of these methods via numerical experiments and we conclude in Section 5.

2 Gyrokinetic field equations in particle simulation model

In this section, we present a classification of the elliptic type electromagnetic field equations that appear in gyrokinetic particle simulations. The classification is given in terms of α in Eq. (1.1).

First, we discuss the gyrokinetic Poisson equation which solves for the electrostatic fluctuation Φ. The general form of the gyrokinetic Poisson equation is in an integral form [30,34] but reduces to the form of Eq. (1.1) in a short wave length limit. Following Lee [30], the gyrokinetic Poisson equation is given by

$$-\frac{\tau}{\lambda_d^2}\left(\Phi-\tilde{\Phi}\right)=-4\pi e\left(\delta\bar{n}_i-\delta n_e\right), \tag{2.1}$$

where e is the unit charge, λ_d is the electron Debye length, $\delta\bar{n}_i$, δn_e are the fluctuation part of the ion and the electron guiding center charge density, and $\tau = T_e/T_i$ is the ratio between the equilibrium electron temperature T_e and the equilibrium ion temperature T_i. Note the Debye shielding term [30,34] is neglected in Eq. (2.1). The $\bar{}$ sign operated on δn_i represents the average over the gyro-phase. In Eq. (2.1), $\tilde{\Phi}$ is the second gyro-phase averaged potential [30]. To solve Eq. (2.1) numerically, an iterative double-gyro-averaging scheme is employed [34]. However, as noted above, the iterative method (under the integral form) cannot be applied for the non-adiabatic kinetic electrons. In the presence of non-adiabatic electrons, the inversion matrix of the iterative method cannot be diagonally dominant [38]. By an expansion in the long wavelength limit, the left side of Eq. (2.1) becomes [30]

$$-\tau\lambda_d^{-2}\left(\Phi-\tilde{\Phi}\right)\sim\tau(\omega_{pi}/\Omega_i)^2\nabla_\perp^2\Phi, \tag{2.2}$$

where ω_{pi} is the ion plasma frequency and Ω_i is the ion cyclotron frequency. Instead, to calculate the response of the short wave length mode correctly, Padé approximation [22] is introduced on the right side of the gyrokinetic Poisson equation. Normalizing Eq. (2.1) with ion gyro-radius for the length, the background density n_0, and T_e/e for the potential, we obtain

$$\nabla_\perp^2\Phi=-\left(1-\frac{1}{\tau}\nabla_\perp^2\right)\left(\delta\bar{n}_i-\delta n_e\right). \tag{2.3}$$

Compared to Eq. (1.1), $f=\left(1-\tau^{-1}\nabla_\perp^2\right)\left(\delta\bar{n}_i-\delta n_e\right)$ and $\alpha=0$. The adiabatic electron limit ($\delta n_e=\Phi$) corresponds to $\alpha=1$ in Eq. (1.1).

Second, Ampere's law for the magnetic perturbation is solved for the magnetic fluctuation. Remembering the magnetic field **B** is given by the curl of vector potential **A**, the magnetic perturbation is given by the parallel vector potential $A_\parallel$ [14]

$$\nabla^2 A=-J_\parallel, \tag{2.4}$$

where $J_\parallel$ is the plasma current obtained from the particles. Here, the subscripts $\parallel$ denote the direction parallel to the equilibrium magnetic field. Equation (2.4) is also an elliptic partial differential equation with $\alpha=0$ in Eq. (1.1). If we restrict ourselves to the short wave length micro-instabilities ($k_\perp^2 \gg k_\parallel^2$), we can apply $\nabla^2 \sim \nabla_\perp^2$ and the equation reduces to two dimensional as in the gyrokinetic Poisson equation.

Third, Ohm's law

$$\left(\nabla_\perp^2-\beta\frac{m_i}{m_e}\right)\frac{\partial\Phi}{\partial t}=S, \tag{2.5}$$

is employed in the electromagnetic split-weight scheme [31]. Here, β is the ratio between the plasma pressure and the magnetic pressure, m_e/m_i is the mass ratio between electrons and the ions, and S on the right is the source term calculated from third order moment of the particle velocity. For this latter case, we have $0<\alpha=\beta(m_i/m_e)$ in Eq. (1.1). The α value can be either smaller or larger than unity depending on the β values.

The global gyrokinetic simulation code (GTC, [33]) 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 approximately 100 finer mesh in the toroidal direction). At first glance the GTC grid seems as a conventional FEM grid (on each poloidal plane). However, the grid rotates in the poloidal 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. To retain the accuracy of the finite element method, the labeling of the vertices are changed depending on the poloidal planes. Note that the solver is not restricted to large aspect ratio devices but can still be applied to spherical tori [National Spherical Torus Experiment (NSTX), [39] for example] by casting Eq. (2.3) to the original integral form Eq. (2.1) (averaging over the projection of the gyro-orbit on a non-circular poloidal plane).

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

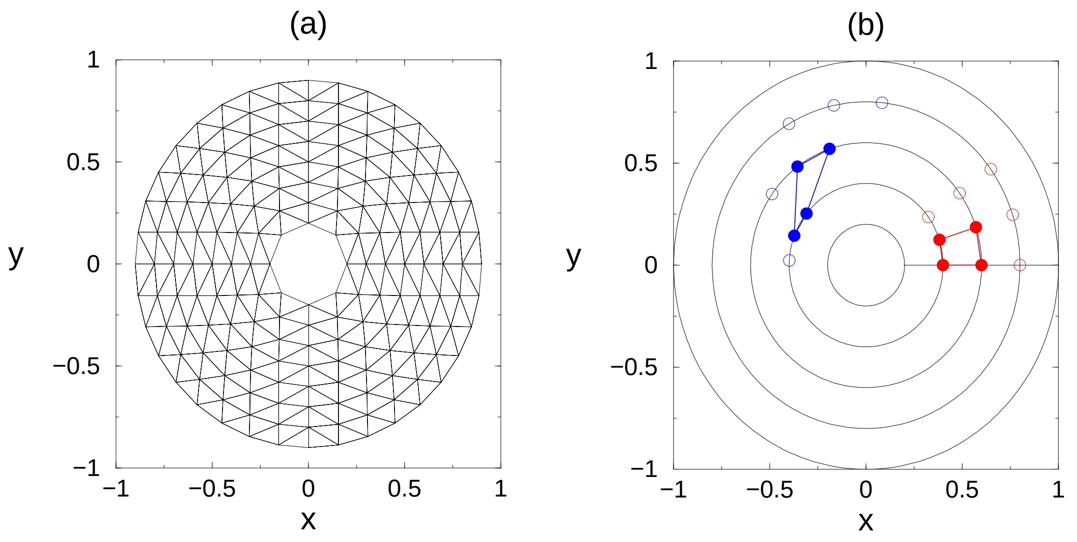

Figure 1: Adaptation of the FEM generator to the GTC grid. Here, very small numbers of grid points are taken to emphasize the GTC grid topology. (a) A conceptual plot of the logically non-rectangular GTC grid. Triangular finite elements with a linear shape functions are adapted. The poloidal plane is 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 that suggests the field alignment. The red blocks illustrate the grid points at one toroidal angle, while the blue blocks illustrate those on the other side of the torus. Note each radius has a different rotation speed (not a rigid rotation) depending on the magnetic field line pitch.

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 is 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 a Galerkin (or variational) coarse grid operator ($A_{coarse} \leftarrow RA_{fine}P$) 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* [41], described in Algorithm 3.1, using a smoother $x \leftarrow S(A,b)$ along with the P and R matrices, on each level.

Algorithm 3.1: Multigrid *V-cycle* Algorithm.

```
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 (pre) smoother
    $r_i \leftarrow b_i - Ax_i$                  – residual calculation
    $r_{i+1} \leftarrow R_{i+1}(r_i)$              – restriction of residual
    $x_{i+1} \leftarrow MGV(R_{i+1}A_iP_{i+1},r_{i+1})$ – recursive call
    $x_i \leftarrow x_i + P_{i+1}(x_{i+1})$         – prolongate 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 (post) smoother
  else
    $x_i \leftarrow A_i^{-1} r_i$                    – direct solve of coarsest grid
  return $x_i$
```

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. Moreover, our matrices are symmetric positive definite and the conjugate gradients (CG) Krylov algorithm is used; CG requires that the preconditioner be symmetric. Symmetry is achieved by setting $R = P^T$ and by using "pre" and "post" smoothing. For instance, if symmetry is not required, then the first two lines of the V-cycle, in Algorithm 3.1, can be removed, resulting in post smoothing only. Additionally, if Gauss-Seidel is used as the smoother it must be symmetrized [5].

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_{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 $R^T = P$, determining the coarse matrix $A_{coarse} = RA_{fine}P$, 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 [24] (Appendix A), and 2) smoothed aggregation [44] (Appendix B).

4 Numerical results

Here we briefly discuss the background of the physics problem and the numerical simulation before investigating the effectiveness of the preconditioners. The Gyrokinetic Toroidal Code (GTC) we employed is designed to study short wave length microinstabilities. GTC is an advanced gyrokinetic simulation codes, and 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. Two cases are considered here: 1) adiabatic electrons [$\delta n_e = u$ is applied in Eq. (2.1) giving rise to a Helmholtz operating on u] and, 2) non-adiabatic electrons; these two cases result in the operator $-\nabla^2 u + \alpha u = f$, with $\alpha = 1$ or $\alpha = 0$ respectively.

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. (1.1) 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 with $0.165 \leq r \leq 0.33m$. The geometry we employ is toroidal, with a circular cross section. Several cases are generated by doubling grid resolution in both the radial and the poloidal directions (thus the number of grid points N per plane increases by a factor of four). Grids of size 38K, 120K, and 640K are formed in this manner and a grid with N=1.2M is generated with $0.0825 \leq r \leq 0.4125$.

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 [32]. The classical algebraic multigrid method is implemented in the HYPRE solver package from Lawrence Livermore National Laboratory [24]. The smoothed aggregation preconditioner is implemented in the package *Prometheus* from Columbia University [1]. Prometheus is built on the parallel numerical package *PETSc* [7], and the mesh partitioner Parmetis [28]. 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. All four packages employed are supported by the Terascale Optimal PDE Simulators (TOPS, http://www.tops-scidac.org) of the U.S. DOE SciDAC initiative.

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 [2]. 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 properties 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 (see Algorithm 3.1) 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 [4]. 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 weak 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 enough processors (from 1 to 32) to keep about 38,000 (38K) equations per processor. To conserve computer resources, these problems were run for 20 time steps, which is about a factor of 100 times fewer that would be run for a full simulation, and with only 8 poloidal planes instead of the usual 64. There is no significant difference expected in terms of challenges for the linear solvers between these performance runs and production physics runs. For each time step, one system of the type under investigation (i.e., Helmholtz or Poisson) is solved to a relative residual tolerance of 1.0^{-6}, that is, convergence is declared when the two norm of the residual r ($r \equiv b - A\hat{x}$) is less than $1.0^{-6}\|b\|_2$. Note that the direct solvers provide a much smaller tolerance, about $1.0^{-14}\|b\|_2$, 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

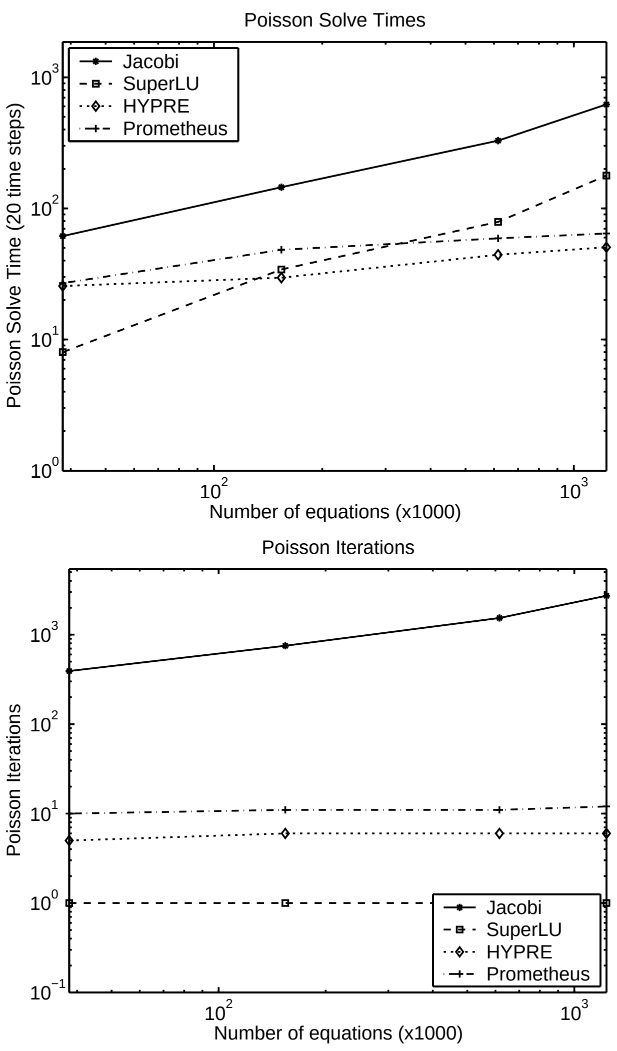

Figure 2: Pure Poisson ($\alpha = 0$).

multigrid methods and about $\mathcal{O}(N^2)$ 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 the performance of linear solvers on the pure Poisson problem, that is with $\alpha = 0$ in Eq. (1.1). Fig. 2 (top) shows the solution times for four cases, ranging in size from 38K equation to 1.2M equations. Fig. 2 (bottom) show the iteration count vs. number of equations. 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 only 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 $\mathcal{O}(N^{3/2})$. The iteration counts, and hence the solve times, for the Jacobi method should, according to theory, go up like $\mathcal{O}(N^{3/2})$. We do indeed see growth of this approximate order.

4.4 Helmholtz problem

This section investigates the performance of methods on the Helmholtz problem, that is with $\alpha = 1.0$ in Eq. (1.1). Fig. 3 (top) shows the solve times for four cases going from 38K equation to 1.2M equations, and Fig. 2 (bottom) show the iteration count vs. number of equations. This 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. For analysis of this phenomenon on an idealized geometry amenable to Fourier decomposition, see [18]. 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 $\mathcal{O}(N^{3/2})$.

4.5 Total times

This section investigates the 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:

1. *Smooth*: Filter high frequencies (not parallelized).
2. *Field*: Compute the gradient of the potential for the electric field (not parallelized).
3. *Charge*: Deposit charge onto the finite element mesh.
4. *Solve*: Solve for the potential - the focus of this paper.
5. *Push*: Push the particles in the electric field.

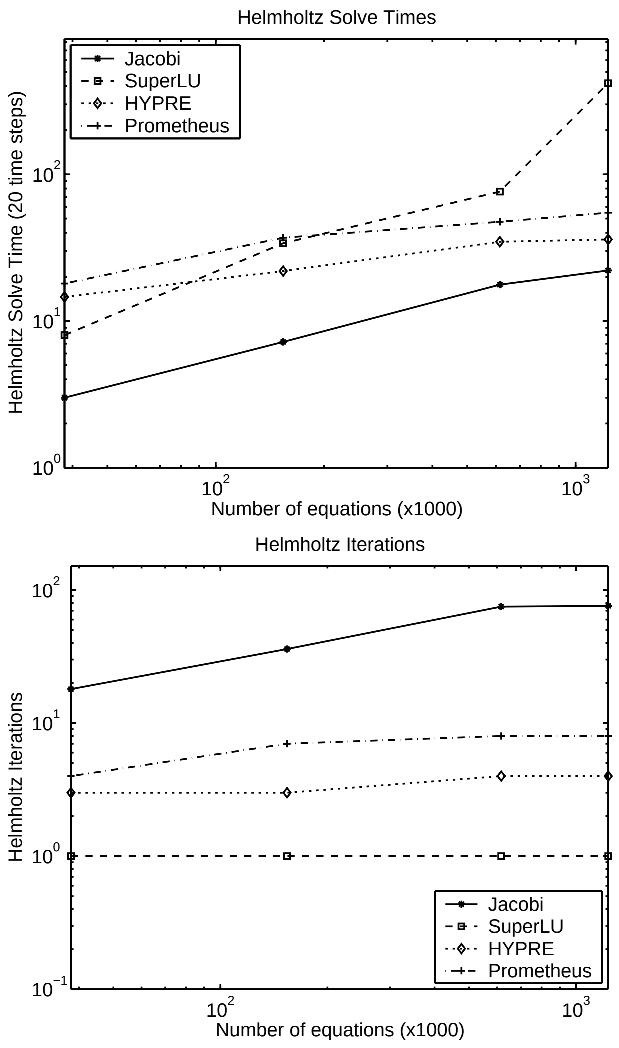

Figure 3: Helmholtz ($\alpha = 1.0$).

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.

Fig. 4 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 Fig. 4. The SP does exhibit nonrepeatable performance

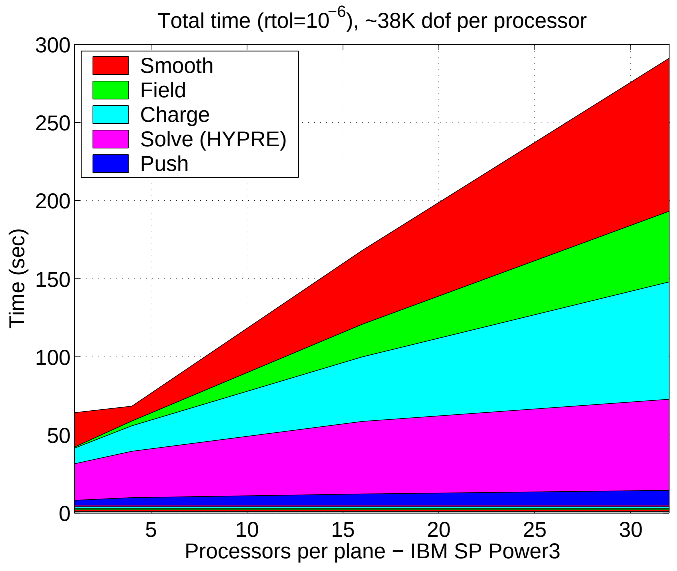

Figure 4: Total times.

and there could be some load imbalance in other parts of the code that accumulated in the synchronization 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 [3] 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). Applications to shaped plasma (NSTX, [39] 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 Department of Energy under contract FG02-04ER25652, Cooperative Agreement No. DE-FC02-04ER54796 and in part by SciDAC Center for Gyrokinetic Particle Simulation of Turbulent Transport in Burning Plasmas.

A Classical algebraic multigrid

What is known as "classical algebraic multigrid" was introduced in the 1980s [40], and methods of this type are available in the HYPRE linear solver library and are used in the numerical experiments in this study [24]. Classical AMG fits into the standard AMG framework described in Section 3 and as such only the interpolation operator P need be described to define the method. The smoothers $S^{\nu 1}(A,b)$ are fairly standard and are described in the numerical experiments in Section 4. This discussion follows that of Henson [24].

Coarse-grid space construction in classical AMG proceeds by partitioning the node set Ω of the graph of the matrix on any given grid into two sets: F-points and C-points. Heuristic rules are used to chose the C-points, which will be used to define one coarse grid function, so that all F-points are interpolated by a set of C-points. These C-points, or coarse-grid points, associated with an unknown u_i are used to represent, or interpolate, a nearby unknown u_j. A point i is said to *depend* on a point j if the value of the unknown u_j is "important" in determining the value of u_i. In this case it is said that *j influences i*. This set S_i is the set of points on which a point i depends. The definition used in classical AMG is:

$$S_i \equiv \left\{ j \neq i : -a_{ij} \geq \alpha \max_{k \neq i} (-a_{ik}) \right\}, \tag{A.1}$$

with α typically set to $\alpha = 0.25$, where a_{ij} is the j^{th} element of the i^{th} row of the matrix A. Also define the set $S_i^T \equiv \{ j : i \in S_j \}$, that is the set of j points that are influenced by i. Two heuristics are employed to increase the quality of the coarse-grid spaces:

C1: For each $i \in F$, each $j \in S_i$ is either in C or $S_j \cap C_i \neq \emptyset$

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

C1 is intended to insure that the value of u_j is represented in the interpolation formula for u_i if i strongly depends on j, even when j is not a C point. **C2** is intended to limit the size of the coarse grid and hence to maintain reasonable computational complexity for the coarse grids. It is not possible, in general, to satisfy **C1** and **C2** simultaneously; the HYPRE library enforces **C1** exactly at the expense of **C2** when necessary.

Given a set of F-points the classical AMG algorithm constructs the prolongation operator as follows. First the prolongation operator for the C-points is simply the identity matrix, that is,

$$P_{ij} = \begin{cases} 1.0 & \text{if } i=j \\ 0.0 & \text{otherwise} \end{cases} \tag{A.2}$$

if the C-points are ordered first. This defines the rows of P that are associated with the C-points. The rows of P that are associated with the F-point are computed by first defining a partitioning of the nodes that an F-point i is connected to in the graph of the matrix into three disjoint set: 1) C_i as defined above, 2) D_i^s, the points that influence i but are not coarse interpolation points and 3) D_i^w, the rest of the neighbors of i (i.e., nodes that do not influence i). The basic definition of the classical AMG interpolation operator is for all $i \in F$-points:

$$P_{ij} = -\frac{1}{a_{ij} + \sum_{k\in D_i^w} a_{ik}} \left(a_{ij} + \sum_{k\in D_i^s} \frac{a_{ik}a_{kj}}{\sum_{m\in C_i} a_{km}} \right)$$

and for all $i \in C$-points use Eq. (A.2). This defines the classical AMG method along with the smoother discussed in Section 4. This algorithm is on firm mathematical ground only for M-matrices, which we have because M3D uses linear finite element spaces.

B Smoothed aggregation multigrid

Aggregation multigrid methods require the null space or kernel of the operator (without Dirichlet boundary conditions applied). For the Poisson operators considered here this is simply the constant vector (e.g., a vector of all ones). Many *plane* aggregation methods have been developed [11,19,20]. 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 below and [44]).

An important improvement in plane aggregation methods is the addition of smoothing which provides significant improvement to the convergence bounds of these methods [35,44], and is especially effective in practice on more challenging problems [2].

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 corresponding to the constant vector):

1. Construct aggregates (a nodal partitioning) on the current (fine) grid i, as described below.
2. For each aggregate J extract the submatrix B_i^J of B_i associated with the nodes in aggregate J.

3. On each aggregate J construct the initial prolongator $\bar{P}_i^J$ with a QR factorization: $B_i^J \rightarrow \bar{P}_i^J B_{i+1}^J$.
4. B_{i+1}^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.
5. 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 $\bar{P}_i$.
6. Each column of $\bar{P}_i$ is "smoothed" with one iteration of an iterative method to provide the prolongator P_i for grid i: $P_i \leftarrow (I - \tau D_i^{-1} A_i)\bar{P}_i$.
7. 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 (i.e., 1 by 1 diagonal blocks of A) is used in this paper and $\tau = 1.5/\lambda_i$ where λ_i is an estimate of the highest eigenvalue of $D_i^{-1}A_i$ (see [35] for details). The construction of the aggregates in step 1 above is the last item in the algorithm that remains to be specified.

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 effect the convergence rate and the memory complexity of the coarse grids which significantly effects the complexity of each iteration and the setup cost (i.e., coarse grid construction). We employ a maximal independent set (MIS) with post-processing similar to the original algorithm proposed by Vaněk et al. [44]. Vaněk 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 [17, 21, 36]). 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 weights w_{ij} between two nodes i and j are computed with $w_{ij} = \|a_{ij}\| / \sqrt{\|a_{ii}\|\,\|a_{jj}\|}$, where a_{ij} is the d by d submatrix 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, i.e., 1 for our test problems). Note that for symmetric positive definite problems $w_{ij} \leq 1.0$ if the two–norm $\|a_{ij}\|_2$ is used; here for computational simplicity we use an average of the one and infinity norms $\|a_{ij}\| = (\|a_{ij}\|_1 + \|a_{ij}\|_\infty)/2$. The MIS is computed with a graph that has been modified by dropping edges that have weights that fall below a certain threshold ϵ; we use $\epsilon = 0.08 \cdot 2^{1-l}$, where l is the grid number as suggested by Vaněk [44].

Common "greedy" MIS algorithms naturally construct a nodal partitioning; these partitions, however, tend to be too large on 3D problems (i.e., 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 size be less than two times the minimum degree of the nodes in the original aggregate. This heuristic is used to limit the size of aggregates because large aggregates would create a "bottleneck" in the convergence of the solver in that the error is relatively poorly resolved by the coarse grid correction on 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.

References

[1] M. F. Adams, Prometheus, www.columbia.edu/$\sim$ma2325.

[2] M. F. Adams, Evaluation of three unstructured multigrid methods on 3D finite element problems in solid mechanics, Int. J. Numer. Meth. Engrg., 55 (2002), 519-534.

[3] 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.

[4] M. F. Adams, M. Brezina, J. J. Hu and R. S. Tuminaro, Parallel multigrid smoothing: polynomial versus Gauss–Seidel, J. Comput. Phys., 188 (2003), 593-610.

[5] M. F. Adams, Distributed memory unstructured Gauss–Seidel slgorithm for multigrid smoothers, in: ACM/IEEE Proceedings of SC2001: High Performance Networking and Computing, 2001.

[6] H. Alfven, Existence of electromagnetic-hydrodynamic waves, Nature 150 (1942), 405.

[7] S. Balay, W. D. Gropp, L. C. McInnes and B. F. Smith, PETSc 2.0 users manual, tech. rep., Argonne National Laboratory, 1996.

[8] D. Braess, On the combination of the multigrid method and conjugate gradients, in: W. Hackbusch and U. Trottenberg (Eds.), Multigrid Methods II, Springer-Verlag, Berlin, 1986, pp. 52-64.

[9] A. Brandt, Multi-level adaptive solutions to boundary value problems, Math. Comput., 31 (1977), 333-390.

[10] W. L. Briggs, V. E. Henson and S. F. McCormick, Multigrid Tutorial, 2nd ed., SIAM, Philadelphia, 2000.

[11] V. E. Bulgakov and G. Kuhn, High-performance multilevel iterative aggregation solver for large finite-element structural analysis problems, Int. J. Numer. Meth. Engrg., 38 (1995), 3529-3544.

[12] 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), 1167-1181.

[13] C. Z. Cheng and G. Knorr, Integration of Vlasov equation in configuration space, J. Comput. Phys., 22 (1976), 330-351.

[14] C. G. Darwin, The dynamical motion of charged particles, Philos. Mag. 39 (1920), 537.
[15] J. Dawson, One dimensional plasma model, Phys. Fluids, 5 (1962), 445-459.
[16] F. F. Chen, Introduction to Plasma Physics and Controlled Fusion, Plenum Press, New York, 1983, pp. 218.
[17] J. Dendy, M. Ida and J. Rutledge, A semicoarsening multigrid algorithm for SIMD machines, SIAM J. Sci. Stat. Comput., 13 (1992), 1460-1469.
[18] A. Ern, V. Giovangigli, D. E. Keyes and M. D. Smooke, Towards polyalgorithmic linear system solvers for nonlinear elliptic systems, SIAM J. Sci. Comput. , 15 (1994), 681-703.
[19] C. Farhat and F.-X. Roux, A method of finite element tearing and interconnecting and its parallel solution algorithm, Int. J. Numer. Meth. Engrg., 32 (1991), 1205-1227.
[20] J. Fish and V. Belsky, Generalized aggregation multilevel solver, Int. J. Numer. Meth. Engrg., 40 (1997), 4341-4361.
[21] W. Hackbusch, Multi-grid Methods and Applications, Springer-Verlag, Berlin, 1985.
[22] T. S. Hahm and L. Chen, Theory of semicollisional kinetic Alfven modes in sheared magnetic-fields, Phys. Fluids, 28 (1985), 3061.
[23] 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), 3019-3022.
[24] V. Henson and U. Yang, BoomerAMG: A parallel algebraic multigrid solver and preconditioner, Tech. Rep. UCRL-JC-141495, Lawrence Livermore National Laboratory, 2000.
[25] M. R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Natl. Bur. Stand., 49 (1952), 409-436.
[26] N. J. Higham, Accuracy and Stability of Numerical Algorithms, 2nd ed., SIAM, Philadelphia, 2002.
[27] ITER, www.iter.org.
[28] G. Karypis and V. Kumar, Parallel multilevel k-way partitioning scheme for irregular graphs, in: ACM/IEEE Proceedings of SC1996: High Performance Networking and Computing, 1996.
[29] W. W. Lee, Gyrokinetic approach in particle simulation, Phys. Fluids, 26 (1983), 556-562.
[30] W. W. Lee, Gyrokinetic particle simulation-model, J. Comput. Phys., 72 (1987), 243-269.
[31] W. W. Lee, J. L. V. Lewandwoski, T. S. Hahm and Z. Lin, Shear-Alfven waves in gyrokinetic plasmas, Phys. Plasmas, 8 (2001), 4435-4400.
[32] X. S. Li and J. W. Demmel, SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems, ACM T. Math. Software, 29 (2003), 110-140.
[33] Z. Lin, T. S. Hahm, W. W. Lee, W. M. Tang and R. B. White, Turbulent transport reduction by zonal flows: Massively parallel simulations, Science, 281 (1998), 1835-1837.
[34] Z. Lin and W. W. Lee, Method for solving the gyrokinetic Poisson equation in general geometry, Phys. Rev. E, 52 (1995), 5646-5652.
[35] J. Mandel, M. Brezina and P. Vaněk, Energy optimization of algebraic multigrid bases, Tech. Rep. 125, UCD/CCM, February 1998.
[36] D. Mavriplis, Directional agglomeration multigrid techniques for high-Reynolds number viscous flows, AIAA J., 37 (1999), 1222-1230.
[37] Y. Nishmura, Z. Lin, J. Lewandowski and S. Ethier, A finite element Poisson solver for gyrokinetic particle simulations in a global field aligned mesh, J. Comput. Phys., 214 (2006), 657-671.
[38] Y. Nishmura and Z. Lin, A finite element mesh in a tokamak edge, Contrib. Plasm. Phys., 46 (2006), 551-556.
[39] Y. K. M. Peng, The physics of spherical torus plasmas, Phys. Plasmas, 7 (2000), 1681-1692.

[40] J. W. Ruge and K. Stüben, Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG), in: D. J. Paddon and H. Holstein (Eds.), Multigrid Methods for Integral and Differential Equations, The Institute of Mathematics and its Applications Conference Series, Clarendon Press, Oxford, 1985, pp. 169-212.
[41] B. Smith, P. Bjorstad and W. Gropp, Domain Decomposition, Cambridge University Press, 1996.
[42] C. R. Sovinec, Private communications, 2004.
[43] U. Trottenberg, C. Oosterlee and A. Schüller, Multigrid, Academic Press, London, 2001.
[44] 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.