Journal of Computational Physics **175,** 249–268 (2002)
doi:10.1006/jcph.2001.6944, available online at http://www.idealibrary.com on IDEAL®

Application of Parallel Implicit Methods to Edge-Plasma Numerical Simulations

T. D. Rognlien, X. Q. Xu, and A. C. Hindmarsh

Lawrence Livermore National Laboratory, L-630, P.O. Box 808, Livermore, California 94551
E-mail: trognlien@llnl.gov

Received February 2, 2001; revised September 14, 2001

A description is given of the parallelization algorithms and results for two codes used extensively to model edge plasmas in magnetic fusion energy devices. The codes are UEDGE, which calculates two-dimensional plasma and neutral gas profiles over long equilibrium time scales, and BOUT, which calculates three-dimensional plasma turbulence using experimental or UEDGE profiles. Both codes describe the plasma behavior using fluid equations. A domain decomposition model is used for parallelization by dividing the global spatial simulation region into a set of domains. This approach allows the use of a recently developed Newton–Krylov numerical solver, PVODE. Results show nearly an order of magnitude speedup in execution time for the plasma transport equations with UEDGE when the time-dependent system is integrated to steady state. A limitation that is identified for UEDGE is the inclusion of the (unmagnetized) fluid gas equations on a highly anisotropic mesh. The speedup of BOUT scales nearly linearly up to 64 processors and gets an additional speedup factor of 3–6 by using the fully implicit Newton–Krylov solver compared to an Adams predictor corrector. The turbulent transport coefficients obtained from BOUT guide the use of anomalous transport models within UEDGE, with the eventual goal of a self-consistent coupling. © 2002 Elsevier Science

Key Words: parallel computation; edge plasma; plasma transport; plasma turbulence; Newton–Krylov.

1. INTRODUCTION

The goal of this work is to develop numerical codes for simulation of edge plasmas for magnetic fusion energy (MFE) devices that exploit the power of parallel computers. Understanding edge plasmas is central to the problems of high heat flux from plasma power exhaust, adequate helium-ash removal, and sufficient edge temperature to reduce turbulent core transport, all recognized as critical issues for magnetic fusion reactors. Proper assessment of these issues requires detailed computer codes. Over the last several years, we have witnessed a period of unprecedented growth in the computer power available for modeling physical problems. To utilize this computational power, one needs to make the shift from coding for single processors to coding for multiprocessor computers. For codes with explicit time advancement, this shift is relatively straightforward because only neighboring quantities enter expressions for the solution at each time step. However, many physical problems, including fusion edge plasmas, contain various phenomena that yield a wide range of time scales that render their descriptive equations "stiff" in the numerical sense. Here, implicit methods are especially useful in achieving acceptably large time steps, but implicit methods require solution of large algebraic systems and thus present a bigger challenge for parallelization compared to explicit methods.

The simulation of edge plasmas has many time scales because of the simultaneous modeling of ion and electron transport along and across a confining magnetic field, together with neutral particle processes. Here we describe the parallelization of two codes that simulate the edge-plasma region in tokamak devices: UEDGE [1] solves for the slowly evolving two-dimensional (2-D) profiles of a multispecies plasma and neutrals given some anomalous cross-field diffusion coefficients, and BOUT [2] solves for the three-dimensional (3-D) turbulence that gives rise to the anomalous diffusion. These two codes are thus complementary in solving different aspects of the edge-plasma transport problem; UEDGE needs BOUT's turbulent transport results, while BOUT needs UEDGE's plasma profiles. Each code can take from a day to weeks on single-processor computers for large problems with the longer times being for the turbulence simulations. An essential step to coupling these calculations is speeding up their individual execution times since many iterations may be needed for a consistent solution.

This parallelization work benefits from the development of two related parallel implicit solvers by Hindmarsh and Taylor [3]: PVODE solves a system of time-dependent ordinary differential equations (ODEs), and KINSOL solves a system of nonlinear equations typically aimed at finding steady-state solutions. In this paper, we only consider the PVODE solver. The serial versions of these Newton–Krylov solvers, VODPK [4] and NKSOL [5], have been used very productively for the serial version of UEDGE. However, our experience [1, 6] and that of others [7] on serial computers shows that for such solvers to work well for UEDGE, this strongly nonlinear system of equations must be well preconditioned. The aim of the preconditioning step is to solve a closely related problem in a more efficient but approximate manner, thereby yielding a more easily solved problem for the Krylov method [4, 5]. Thus, part of our work for UEDGE focuses on development of a parallel preconditioner based on a domain decomposition model [8]. This Fortran preconditioner is interfaced to the C-language solvers PVODE and KINSOL [3] on a variety of parallel computer platforms ranging from the massively parallel T3E computer to shared-memory workstations with multiple processors (SUN and DEC). This aspect of our work demonstrates, for a complex problem, how one can reuse existing Fortran coding and, with moderate extensions, utilize recently developed implicit parallel solvers.

Another element of portability is provided by implementing message passing between multiple processors by using the message-passing interface (MPI) package [9]. This package is available on many different platforms and allows one to utilize either shared memory or individual processor memory without changing the code. With MPI, one can now use processors on one computer or a network of computers.

The impact of domain decomposition on Newton–Krylov preconditioning methods for the 2-D plasma transport equations has been studied by Knoll *et al.* [10] using a serial computer (one processor). Our UEDGE work is related but differs in that we solve the domain-decomposed system on a parallel computer, include the full multiply connected toroidal geometry with coupling between two internal boundaries rather than a slab geometry, and focus on a time-dependent solution method with expanding time step to reach steady state. We consider the speedup of the complete nonlinear calculation, while Ref. [10] compares the effectiveness of different domain blocking strategies on the inner iteration count for the Krylov solver. Also, the domain-decomposition work in Ref. [10] uses the transpose-free QMR algorithm for the inner iteration loop, whereas we always use the generalized minimum residual (GMRES) method. We shall show that our specific examples are dominated by the Newton–Krylov work and only secondarily by the formation of the preconditioning matrix, P, while the examples in Ref. [10] are dominated by the formation of P. We find a similar degradation of the quality of the domain-decomposed preconditioner owing to the loss of information by omitting coupling between the domains as reported in Ref. [10]. For the work presented here, the primary gain in "wall-clock" execution time arises from the parallelization, which is degraded somewhat by the domain preconditioner.

The 3-D BOUT code is parallelized using the same general domain-decomposition model as UEDGE, but our emphasis for BOUT is somewhat different. First, since BOUT must follow the rapid fluctuations of the turbulent fields, we compare two fully implicit schemes for advancing the equations in time and show how the scheme using the Newton–Krylov method is much superior to a predictor–corrector method. Second, we find that the implicit BOUT works well without a preconditioner in that only $\sim$6 Krylov vectors per Newton iteration are needed. This latter fact leads to the straightforward parallelization of BOUT with a nearly linear speedup in computational time with the number of CPU processors. In contrast to UEDGE, which often take very large time steps to find an equilibrium and needs a preconditioner, the good performance of BOUT without a preconditioner is related to the smaller time step required to resolve the turbulent fluctuations. Like UEDGE, the parallel version of BOUT uses MPI for portability.

The plan of the paper is as follows: In Section 2, the geometry and basic equations for the edge-plasma problem are given. The domain-decomposition model used for UEDGE and BOUT is also described in Section 2. The results for the UEDGE parallelization are shown in Section 3. The BOUT results from the fully implicit solver and parallelization are given in Section 4. The conclusions are summarized in Section 5.

2. EQUATIONS, GEOMETRY, AND ALGORITHMS

2.1. *Basic Equations*

The basic models in the UEDGE and BOUT codes begin with the plasma fluid equations of continuity, momentum, and thermal energy for both the electrons and ions in the form given by Braginskii [13]. The continuity equations have the form

$$\frac{\partial n}{\partial t} + \nabla \cdot (n_{e,i}\mathbf{v}_{e,i}) = S^{p}_{e,i}, \tag{1}$$

where $n_{e,i}$ and $\mathbf{v}_{e,i}$ are the electron and ion densities and mean velocities, respectively. The source term $S^{p}_{e,i}$ arises from ionization of neutral gas and recombination.

The momentum equations are given by

$$nm_{e,i}\frac{\partial \mathbf{v}_{e,i}}{\partial t} + m_{e,i}n_{e,i}\mathbf{v}_{e,i}\cdot\nabla\mathbf{v}_{e,i} = -\nabla p_{e,i} + qn_{e,i}(\mathbf{E} + \mathbf{v}_{e,i}\times\mathbf{B}/c) - \nabla\cdot\mathbf{\Pi}_{e,i} - \mathbf{R}_{e,i} + \mathbf{S}^m_{e,i}, \tag{2}$$

where $m_{e,i}$ are the masses, $p_{e,i} = n_{e,i}T_{e,i}$ are the pressures with $T_{e,i}$ being the temperatures, q is the particle charge, $\mathbf{E}$ is the electric field, $\mathbf{B}$ is the magnetic field, c is the speed of light, $\mathbf{\Pi}_{e,i}$ are the viscous tensors, and $\mathbf{R}_{e,i}$ are the thermal forces [13]. The source $\mathbf{S}^m_{i,e}$ contains a sink term $-nm_{i,e}\mathbf{v}_i S^p_{i,e}$ that arises if newly created particles have no drift motion.

The ion and electron energy equations can be written in the form

$$\frac{3}{2}n\frac{\partial T_{e,i}}{\partial t} + \frac{3}{2}n\mathbf{v}_{e,i}\cdot\nabla T_{e,i} + p_{e,i}\nabla\cdot\mathbf{v}_{e,i} = -\nabla\cdot\mathbf{q}_{e,i} - \mathbf{\Pi}_{e,i}\cdot\nabla\mathbf{v}_{e,i} + Q_{e,i}, \tag{3}$$

where $\mathbf{q}_{e,i}$ are the heat fluxes and $Q_{e,i}$ are the volume heating terms [13].

In their general three-dimensional form, the 6 plasma equations given above represent 10 separate partial differential equations for n_e, n_i, $\mathbf{v}_e$, $\mathbf{v}_i$, T_e, and T_i. UEDGE and BOUT use somewhat different assumptions to reduce the complexity of their models. Both codes use the classical parallel transport given by Braginskii, but UEDGE uses enhanced perpendicular transport coefficients to model the effect of the turbulence calculated by BOUT. Here parallel and perpendicular refer to directions relative to the magnetic field, $\mathbf{B}$. In the calculation of the electrostatic potential, ϕ, both codes use the quasineutral condition, $n_e = n_i$, and in addition, BOUT solves for the parallel magnetic vector potential, $A_\parallel$. UEDGE assumes symmetry in a third (toroidal) dimension, whereas BOUT allows fluctuations to have variations in all three spatial dimensions even though its equilibrium profiles are toroidally symmetric.

UEDGE also includes the capability of evolving neutral gas species for each type of ion, which can be described by models of varying sophistication. The simplest neutral description is a diffusion model, where the neutral density, n_n, obeys the continuity equation

$$\frac{\partial}{\partial t}n_n + \frac{\partial}{\partial x}(n_n v_{nx}) + \frac{\partial}{\partial y}(n_n v_{ny}) = (\langle\sigma_r v_e\rangle - \langle\sigma_i v_e\rangle)n_e n_n. \tag{4}$$

Here the neutral velocities, $v_{nx,ny}$, are taken from a diffusion approximation using the charge-exchange collision frequency between ions and neutrals, and the source and sink terms on the right-hand side represent recombination and ionization with rate coefficients $\langle\sigma_r v_e\rangle$ and $\langle\sigma_i v_e\rangle$, respectively. We find that the parallelization of this neutral gas equation on the highly anisotropic mesh performs more poorly than the plasma equations, a point we will return to in Section 3.2.

For the 3-D BOUT turbulence simulations, two auxiliary variables and related equations are introduced to help solve the system of plasma equations [2]. These variables are the parallel current, $j_\parallel$, and vorticity, ϖ, and the related equations are Poisson-like equations for ϕ and $A_\parallel$:

$$\nabla^2_\perp\phi = \varpi \tag{5}$$

$$\nabla^2_\perp A_\parallel = -\frac{4\pi}{c}j_\parallel. \tag{6}$$

The ϕ potential equation is not obtained from Poisson's equation, but rather from the quasineutrality condition and the current continuity equation. Here $\nabla^2_\perp$ refers to the

Laplacian operator in the directions perpendicular to the magnetic field. The solutions to these simple-looking equations, Eqs. (5) and (6), have important consequences for the parallel version of BOUT.

2.2. *Geometry*

Both UEDGE and BOUT are written in general coordinates that can be adopted to a slab, cylinder, or torus by the use of the appropriate metric coefficients. For MFE devices, interest has focused on toroidal devices with an emphasis on tokamaks [14]. The region occupied by the edge plasma for the poloidal plane of a tokamak with a single-null divertor is shown in Fig. 1. The long, sometimes closed lines of the mesh represent poloidal magnetic flux surfaces in which the magnetic field vector lies. For tokamaks, the strongest magnetic field component is in the toroidal direction, out of the plane of the figure.

In the poloidal plane, UEDGE and BOUT use the poloidal flux surfaces as one spatial coordinate, with the second being the curves normal to the flux surfaces shown in Fig. 1a, but a nonorthogonal mesh is sometimes used to conform the mesh to material surfaces at the boundary. For BOUT, toroidal variations are allowed in a segment of the torus as shown in Fig. 1b; this segment is periodically replicated to fill out the torus. Thus, the wavelength of the longest toroidal mode simulated is set by the length of the toroidal segment used.

The numerical discretization schemes used by UEDGE and BOUT are similar in the two dimensions in the poloidal plane. UEDGE uses a conservative finite-volume method and BOUT uses a finite-difference method including a fourth-order spatial discretization for the nonlinear $\mathbf{E} \times \mathbf{B}$ inertial velocity terms.

2.3. *Implicit Algorithms*

The fluid equations solved by both UEDGE and BOUT can be cast in the most general form in terms of a system of ODEs

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u}), \tag{7}$$

where $\mathbf{u}$ is the vector of unknowns, and $\mathbf{f}$ is the result of sources, sinks, and the discretized spatial transport terms. Since UEDGE is usually seeking steady-state solutions, it strives to take the maximum time step (Δt) possible and sometimes works in the limit of $\Delta t \to \infty$. BOUT always follows time dependence, but we wish to do so with optimum efficiency. This section sets the background for understanding the algorithms used for UEDGE and BOUT in Sections 3 and 4.

We consider two implicit schemes, one utilizing a predictor–corrector method and the other using the Newton–Krylov approach. Our codes presently use the Newton–Krylov algorithm utilizing GMRES, but introducing the predictor–corrector method allows us to make a comparison later in Section 4.2. These schemes are exemplified by the Adams method and the backward differentiation formula (BDF) method, respectively. For the Adams method, the advancement of $\mathbf{u}$ from time level $n-1$ to n takes the form

$$\mathbf{u}_n = \mathbf{u}_{n-1} + \Delta t(\alpha_0\mathbf{f}_n + \cdots + \alpha_{k-1}\mathbf{f}_{n-k+1}), \tag{8}$$

where k is the order of the scheme, the α’s are coefficients [15], and $\mathbf{f}_n \equiv \mathbf{f}(\mathbf{u}_n)$. For the BDF method, the advancement is given by

$$\mathbf{u}_n = (\beta_1\mathbf{u}_{n-1} + \cdots + \beta_k\mathbf{u}_{n-k}) + \Delta t\gamma_0\mathbf{f}_n, \tag{9}$$

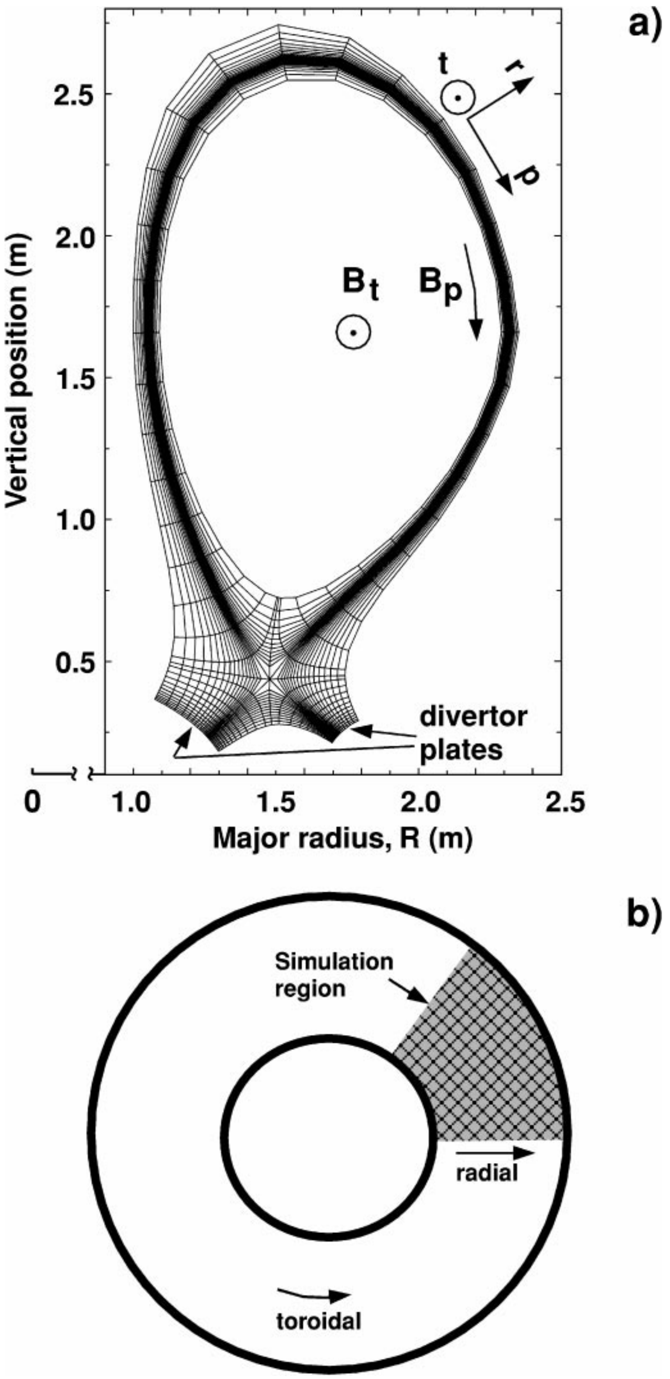


FIG. 1. The toroidal tokamak geometry simulated by the UEDGE and BOUT codes. In (a), the poloidal plane plot shows the 2-D edge region simulated by UEDGE and the mesh used that has one coordinate based on magnetic flux surfaces as provided by an MHD equilibrium code. In addition to simulating the poloidal annulus in (a), BOUT allows fluctuations to have toroidal variations that fit periodically into the toroidal segment shown from the top view in (b). Thus, inclusion on longer toroidal wavelength modes requires using a larger toroidal segment at increased computational cost.

where the β's and γ_0 are coefficients determined by the order (k) used [15]. The Adams method is usually solved by functional iteration; i.e., an approximation to $\mathbf{u}_n$ at iteration j, termed $\mathbf{u}_n^j$, is obtained by evaluating $\mathbf{f}_n$ with $\mathbf{u}_n^{j-1}$; this approach can work well for nonstiff systems. While the BDF method can also use functional iteration, a more effective method is often a Newton iteration that expands $\mathbf{f}_n$ at iteration j as

$$\mathbf{f}(\mathbf{u}^j) \approx \mathbf{f}(\mathbf{u}^{j-1}) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{u}^j - \mathbf{u}^{j-1}). \tag{10}$$

Equation (9) then is a linear equation for $\mathbf{u}_n^j$ that can be written as

$$(\mathbf{I}/\Delta t\gamma_0 - \mathbf{J})\mathbf{u}_n^j = \mathbf{g}, \tag{11}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{J} \equiv \partial\mathbf{f}/\partial\mathbf{u}$ is the Jacobian evaluated with $\mathbf{u}$ from a previous iteratation or time step. Also, $\mathbf{g}$ is a vector that depends on values of $\mathbf{u}$ from the past iteration, $\mathbf{u}^{j-1}$, and at previous time steps as obtained from Eqs. (9 and 10). Equation (11) is usually solved by an iterative method to an accuracy somewhat better than the estimated error in $\mathbf{u}_{n-1}$ from the time advancement; this is known as an inexact Newton method. We shall use a Krylov projection method to solve the linear system [4, 11]. Although more numerical operations are required for such Newton methods per iteration, they often have superior overall performance for stiff ODEs since larger time steps can be used, as we shall illustrate with a BOUT example. Also, UEDGE requires the inexact Newton method for reasonable performance, and it requires that the system of equations be preconditioned.

Newton schemes that utilize a matrix-free Krylov projection method often require preconditioning. The procedure involves solving related linear systems $\mathbf{Pw} = \mathbf{h}$ with a matrix $\mathbf{P}$ that approximates the original matrix but is simpler to solve. By assumption, $\mathbf{P} \sim (\mathbf{I}/\Delta t\gamma_0 - \mathbf{J})$. Noting that $\mathbf{P}^{-1}\mathbf{P} = \mathbf{I}$, we may insert this product into Eq. (11) to form the preconditioned system

$$[(\mathbf{I}/\Delta t\gamma_0 - \mathbf{J})\mathbf{P}^{-1}]\big(\mathbf{P}\mathbf{u}_n^j\big) = \mathbf{g}. \tag{12}$$

The new variables are $\mathbf{P}\mathbf{u}_n$, and this system is easier to solve by iterative methods since $(\mathbf{I}/\Delta t\gamma_0 - \mathbf{J})\mathbf{P}^{-1} \equiv \mathbf{A} \sim \mathbf{I}$ is more diagonally dominant. Each iteration of the Krylov method does require matrix–vector products $\mathbf{Av}$ (with $\mathbf{v}$ being the Krylov basis vector), and these are done using a matrix-free finite-difference quotient approximation to $\mathbf{Jw}$ where $\mathbf{w} = \mathbf{P}^{-1}\mathbf{v}$. More detailed descriptions of the Newton–Krylov algorithm are available from a number of sources, e.g., Refs. [4, 11, 12].

2.4. *Domain-Decomposition Model*

UEDGE and BOUT use the same poloidal mesh, and this region can be divided into domains on parallel computers where separate processors can solve a local problem. However, for the edge-plasma problem, there is a set of natural interior boundaries that need to be identified and accommodated for efficient decomposition. The regions delineated by these interior boundaries are shown in Fig. 2 for both the tokamak poloidal plane and the corresponding mapping to a rectangular domain. Information needs to be passed from cells that touch one of the dotted lines between the private-flux and core regions to the cells along the other dotted line, and vice versa. These interior boundaries are used to account for the periodic boundary conditions used within the core region and the continuity-of-flux condition between the private-flux region 3 and private-flux region 4.

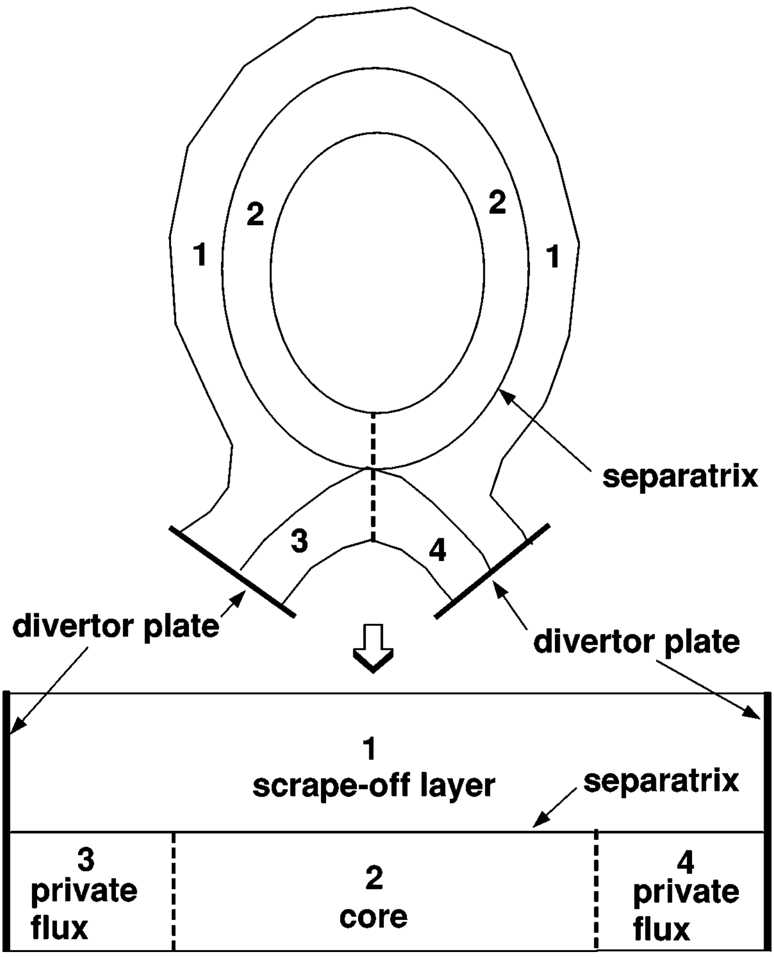


FIG. 2. The poloidal plane is divided into four main regions for the domain decomposition model, each of which can be further subdivided. The four regions are mapped into the rectangular geometry shown in the lower part of the figure by opening the poloidal configuration along the dotted line.

If the selection of the domains is such that the boundaries of major regions 1–4 in Fig. 2 are always included in the boundaries of the domains, then the finite-difference representation in a given domain can be entirely local. Such a domain decomposition is shown in Fig. 3a where 16 domains are used in the poloidal plane.

The information needed to form the local finite-difference approximations to the derivatives at the boundary of the domains is provided by passing the variable data between processors (domains) via MPI [9] to fill the guard cells that surround each domain shown in Fig. 3b. Notice that data needed in a guard cell are not necessarily from the adjacent domain; e.g., the right-side guard-cell data for domain 0 in Fig. 3a come from domain 3. These guard cells do not contain variables that are advanced for each domain, but rather they contain only a copy of this information from other processors. The only exception to this rule is for UEDGE, which uses exterior guard cells to specify boundary conditions; but here the boundary conditions are local, so no message passing is required.

The domain decomposition model plays two roles. First, to utilize the implicit PVODE (or KINSOL) solver [3], we must divide the physical space simulated by our codes in

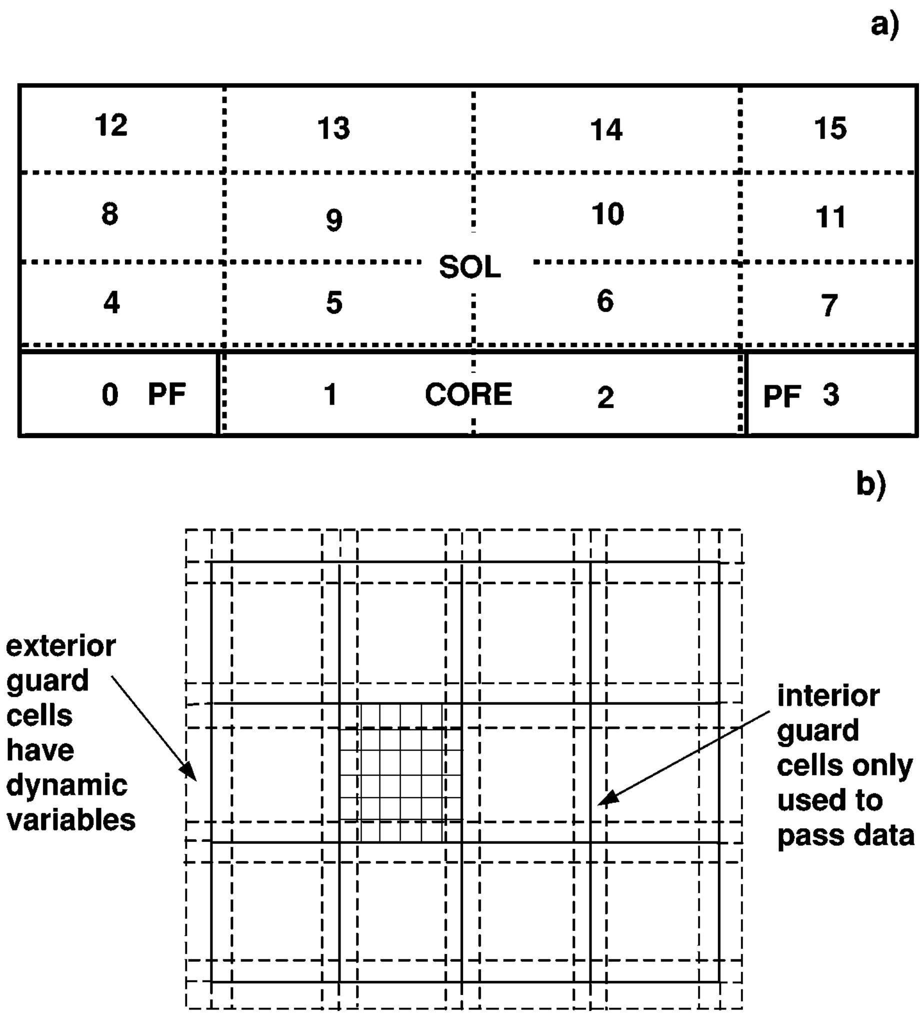

a)
12
13
14
15
8
9
10
11
SOL
4
5
6
7
0 PF
1
CORE
2
PF 3
b)
exterior guard cells have dynamic variables
interior guard cells only used to pass data

FIG. 3. (a) Division of UEDGE geometry into 16 regions is shown. (b) More detail of the mesh is shown within the domains together with the overlapping guard cells.

this manner, where each processor, with guard cells, has all of the information required to evaluate the right-hand side for its domain. The Newton–Krylov solvers, together with modest amount of message passing between domains, allows an implicit solution to the global problem. The second role is that the model provides the basis for the preconditioning algorithm that is required by UEDGE. Here the full set of preconditioner Jacobian elements in P can be efficiently generated in parallel by finite-difference quotients. For efficiency, PVODE reuses P and its factorization for a number of nonlinear iterations. An approximate LU factorization of P on each processor is performed using the ILUT routine [11] with a drop-tolerance parameter of 10^{-3} and a row fill-in parameter of 50. We have also used simpler routines such as ILU(0) [11], but since the LU factorization is not a significant time sink for our problems, we chose to maintain the flexibility of ILUT. Furthermore, since the domain size is typically small compared to full simulation region, one can be more aggressive with LU factorization because it is faster to factor a number m of $n \times n$ matrices than one $mn \times mn$ matrix. The overall procedure used here of not including coupling

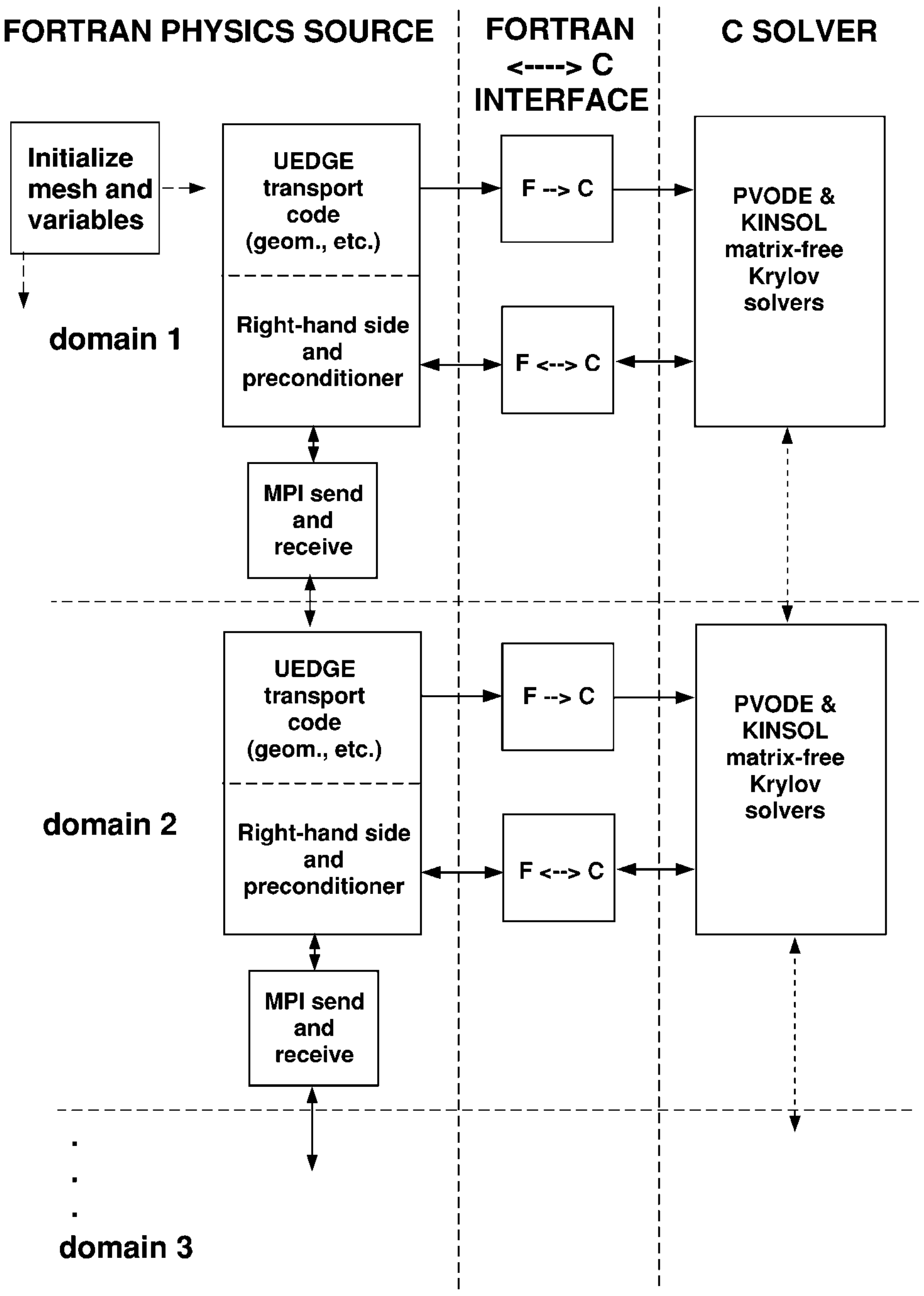

FORTRAN PHYSICS SOURCE
FORTRAN <----> C INTERFACE
C SOLVER
Initialize mesh and variables
UEDGE transport code (geom., etc.)
Right-hand side and preconditioner
F --> C
F <--> C
PVODE & KINSOL matrix-free Krylov solvers
MPI send and receive
domain 1
domain 2
domain 3

FIG. 4. Schematic showing the three major components of the parallel UEDGE code as replicated on each domain or processor.

between domains at the preconditioning level is referred to as additive-Schwarz with zero overlap [12].

The manner in which UEDGE utilizes the PVODE solver can be most succinctly explained by the diagram in Fig. 4. On the left is the main UEDGE calculation of the finite-difference equations that yield the “right-hand side” of the evolutionary equations for each variable. The central column of Fig. 4 shows the wrappers mentioned previously that pass data from the Fortran UEDGE code to the C solvers, and vice versa. Finally, on the right is one of the C solvers, PVODE or KINSOL, which were developed previously [3]. Note that the foregoing model is replicated for all domains or processors. Communication between processors as required to fill guard cells is shown by the “MPI send and receive” boxes in Fig. 4.

The parallel model for BOUT is very similar to that just described for UEDGE, except that BOUT is written in C and thus requires no extra interface routines to utilize the C solvers. Since BOUT must follow the time dependence, only the PVODE solver is used here. Also, as mentioned earlier, BOUT works well without a preconditioner. Some work has been done on testing preconditioners for even more improvement, but more development is needed.

3. IMPLEMENTATION AND RESULTS FOR UEDGE

3.1. *Implementation*

The 2-D plasma transport equations used in UEDGE come from a reduction of those presented in Section 2.1 for the parameters of the edge plasma. This reduction results in five equations for the following variables: ion density, n_i; ion parallel velocity, $v_\parallel$; separate electron and ion temperatures, T_e and T_i; and the electrostatic potential, ϕ. If plasma currents are ignored, as done in the 2-D examples in Section 3.2, the potential becomes a dependent variable, resulting in four basic plasma equations. In addition, impurity species can be included that have their own density and parallel velocities but a common temperature, T_i, with the ions. When the neutral gas is included, at least the neutral continuity equation (4) is solved for each ion isotope.

With UEDGE, we seek efficient steady-state solutions while still retaining the options to simulate time-dependent evolution of the profiles when needed. Using large time steps, or performing nonlinear iterations to steady state with no time step, requires the use of a good preconditioner. Although reduced preconditioners have been tried for this complex problem, we have found that a full preconditioning matrix, P, computed by finite-difference quotients is needed to effectively obtain solutions for a wide range of parameters. This step is distinct from the finite-difference approximation to the matrix–vector product, $\mathbf{Jv}$, used by the PVODE Krylov solver. Such a preconditioner is only updated occasionally during the nonlinear iteration. We have two options for the parallel UEDGE, either using an algorithm developed specifically for UEDGE or, because each region in the domain decomposition model is simply connected yielding band-block-diagonal (BBD) matrices, using the preconditioners PVBBDPRE supplied as part of the PVODE package [3]. The UEDGE-specific algorithm uses a small 2-D “window” that moves across the mesh, providing local Jacobian elements by difference quotients over a restricted range of the right-hand side evaluations to the 2-D window; nonlocal couplings from the multiple-connected regions are built into the algorithm so that it works on either serial or parallel machines. The PVBBDPRE module assumes all couplings are localized to a small neighborhood of a given variable. Here P is a BBD matrix, where each block is generated on one processor and is obtained from a banded difference-quotient approximation. For the domain-decomposed system, the operation counts for the two different methods of calculating P are nearly the same. The importance of frequent updates to P for our problem will be shown in the next section.

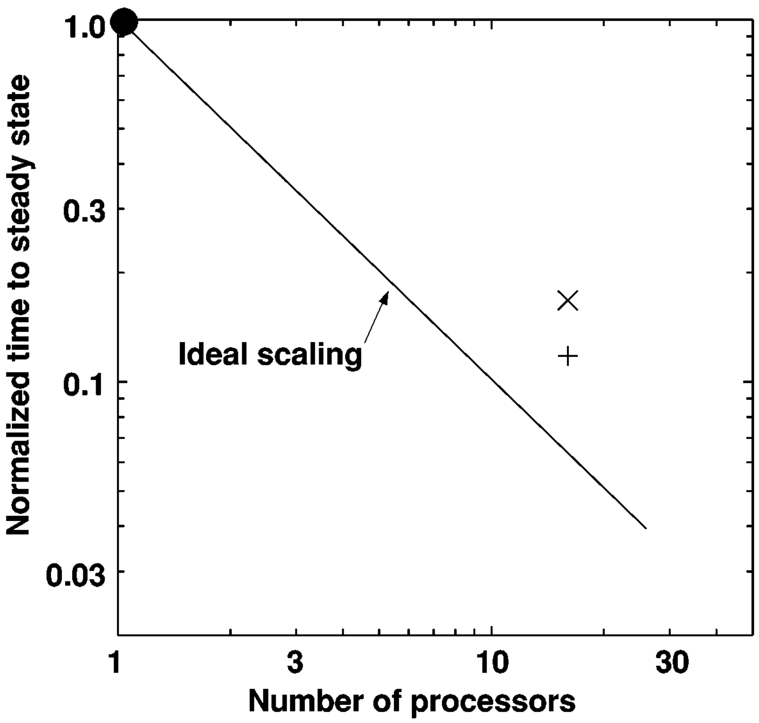
To implement the domain-decomposition model, we have written a routine that automatically divides the global mesh in a manner that respects the "natural" boundaries shown in Fig. 2. One can specify the number of subdomains in each of these regions; any imbalance is handled by assigning fewer equations than the average to a minority of processors. We typically obtain complete load balancing by a proper choice of mesh sizes and number of domains. The routine also sorts through the indexing for the guard cells and provides a map to specify which processors must exchange boundary data. A set of routines was developed that deals with passing data from the master processor to domain processors. These data include the initial guess to the global solution, the global geometrical data, and the mapping index for the guard cells needed for each domain. A similar routine is used to gather the data from all the processors into a global solution at the end of the run. Finally, another set of message-passing routines was constructed to refresh the guard-cell data at the appropriate times during the Jacobian and Newton–Krylov steps.

3.2. *Results*

We run UEDGE on the T3E-600 using the 16 domain configuration shown in Fig. 3 for the full DIII-D tokamak geometry in Fig. 1a. A couple of factors dictate the use of the 16 domain configuration: We need the internal connections between private-flux regions and the two ends of the core region to occur on domain boundaries (see Fig. 2), and the spatial resolution with load balancing requires that we have about three times the number of cells in the SOL as in the core. The computation mesh has 64 poloidal mesh points and 48 radial points so that we can fit the single-domain base case on one processor for direct comparison with the multiprocessor cases.

The input parameters used at the core boundary are $T_{e,i} = 150$ eV, $n_i = 2 \times 10^{19}$ m^{-3}, and zero parallel velocity. The anomalous radial diffusion coefficients are set to 1 m^2 s^{-1}, and the plate particle recycling coefficient is 0.9. The simulation is initialized with a solution obtained for a $T_{e,i} = 100$ eV on the core boundary, and we then measure the computer time required to find the solution when we switch to $T_{e,i} = 150$ eV on the core boundary. The results presented only include evolution of the plasma equations for a fixed neutral background. The execution time normalized to that for one processor is presented in Fig. 5. Here PVODE is used to run to steady state with the plasma equations, and two different preconditioners are used, the case marked $\times$ being PVBBDPRE, and the $+$ point being the internal UEDGE preconditioner. In the table below the figure, the data show the number of function (or right-hand side) evaluations for the Krylov iterations, the number of preconditioner evaluations, the normalized time, and the ideal time. Although the speed of the calculation depends somewhat on the preconditioner used, experience with various approximate preconditioners on serial computers indicates that both work well; errors in the preconditioner typically result in an inability to obtain a solution with UEDGE.

The difference in the speedup results from the two preconditioners in Fig. 5 is due primarily to the frequency with which they are updated. Note from the table in Fig. 5 that the PVBBDPRE case (labeled $\times$) has only about $1/3$ of the preconditioner evaluations compared to the $+$ data point with the UEDGE preconditioner. This difference in the number of preconditioner evaluations is caused by update logic within PVODE that allows the use of the same P even when the time step changes for the internally generated P option but requires a new P when Δt changes for the externally generated P. As a consequence, the $\times$ data point has almost twice the number of overall function evaluations from PVODE. Thus,



Num. PEs	Func. eval	Prec. eval	Nrm. time	Ideal time
1 (●)	2236	60	1.0	1.0
16 (x)	9803	24	0.168	0.0625
16 (+)	5760	67	0.118	0.0625

FIG. 5. Comparison of time to reach a steady-state solution for the parallel UEDGE run on the T3E-600 parallel computer with 1 processor and 16 processors for the plasma equations with PVODE. The point labeled $\times$ uses the PVBBDPRE preconditioner and the $+$ point uses the internal UEDGE preconditioner. The table gives the number of function evaluations, preconditioner evaluations, and the normalized time to steady state.

the results from the two preconditioners indicate the sensitivity of the trade-off between more frequent preconditioner evaluations (and LU factorization) and fewer Newton–Krylov iterations as reflected in the function evaluation count. The one-processor base case uses the UEDGE preconditioner.

A large fraction of the CPU time for all three cases in Fig. 5 arises from two aspects of the computation, namely, the PVODE Newton–Krylov operations and the formation of the preconditioner, P. For the one-processor base case, PVODE uses 58%, the formation of the P's uses 35%, and the remainder is used in the LU factorization/backsolve of P. For the two domain-decomposed parallel cases, the factorization of the smaller P's becomes negligible and the message passing is less than 5%. The PVBBDPRE preconditioner case with 24 P evaluations takes 95% of the remaining time for PVODE and 5% to form the 24 P's. The UEDGE preconditioner case, which has the faster overall time, takes 80% for PVODE and 20% to form the 67 P's. Thus, for our cases, the speedup is largely due to the use of many processors (parallelization), while the less-than-ideal scaling arises from the loss of information from the domain-decomposed preconditioner; more frequent updates of the inexpensive P improves the speed. For other parameters and larger problems, the reduced time required for factorization of P when using many domains can also be important [10].

While it is encouraging to obtain nearly an order of magnitude speedup for the plasma equations in UEDGE, using the relatively simple gas equation shown by Eq. (4) is more difficult. When Eq. (4) is included, a large increase in CPU time is required (>3 times), which can be traced back to the highly anisotropic mesh shown in Fig. 1 [16]. This mesh is chosen to best represent the plasma that flows rapidly along the flux surfaces and transports slowly across the magnetic flux surfaces owing to magnetic confinement. However, the gas evolving from the divertor plates does not experience a magnetic force and is not preferentially confined to the flux surfaces. We have studied this problem in some detail for a simple gas diffusion problem outside the actual tokamak geometry and had the same difficulty. This will be the subject of future research.

4. IMPLEMENTATION AND RESULTS FOR BOUT

4.1. *Implementation*

For edge-plasma turbulence, the application of a fluid model is reasonable in part because of the low temperature and the resulting short mean-free path from Coulomb collisions. While the unstable modes can have wavelengths that are short compared to the scale lengths of equilibrium profiles, the dominant modes have perpendicular wavelengths that are larger than the ion gyroradius, ρ_s, which is consistent with a fluid approach. Thus, it is again appropriate to use the Braginskii fluid equations as presented in Section 2.1. By scaling arguments, we can reduce the full set of fluid equations to a seven-variable set for the electrostatic potential, ϕ; magnetic vector potential, $A_\parallel$; plasma density, n_i; electron and ion temperatures, T_e and T_i; and electron and ion parallel velocities, $v_{e\parallel}$ and $v_{i\parallel}$. Also, the auxiliary variables, $j_\parallel$ and ϖ, obey the potential equations (5 and 6).

To efficiently simulate turbulence with short perpendicular wavelengths compared to parallel wavelengths (i.e., for wavenumbers $k_\parallel \ll k_\perp$), we choose field-line-aligned ballooning coordinates (x, y, z), which are related to the usual flux coordinates [14] (ψ, θ, φ) by the relations $x = \psi - \psi_s$, $y = \theta$, and $z = \varphi - \int \nu(x, y)\, dy$. Here, ψ is the poloidal magnetic flux, θ is the poloidal angle, and φ is the toroidal angle. Also, $\nu \equiv a_e B_t / R B_p$ measures the inverse pitch of the magnetic field line, where a_e is the effective minor radius, R is the major radius, and $B_{t,p}$ are the toroidal and poloidal magnetic fields, respectively. The partial derivatives are $\partial/\partial\psi = \partial/\partial x - \int (\partial\nu/\partial\psi)\, dy \partial/\partial z$, $\partial/\partial\theta = \partial/\partial y - \nu\partial/\partial z$, $\partial/\partial\varphi = \partial/\partial z$, and $\nabla_\parallel = (B_p/a_e B)\partial/\partial y$. The magnetic separatrix is denoted by $\psi = \psi_s$. Here the key ballooning assumption is $|\partial/\partial y| \ll |\nu\partial/\partial z|$ and $\partial/\partial\theta \simeq -\nu\partial/\partial z$. In this choice of coordinates, y, the poloidal angle, is also the coordinate along the field line.

In the most general case, the solution to Eqs. (5 and 6) requires a three-dimensional solver since one of the perpendicular directions is composed of the poloidal and radial components. However, utilization of the ballooning assumption ($\partial/\partial\theta \approx -\nu\partial/\partial z$) with short toroidal wavelengths reduces the potential equations to two dimensions in the radial and toroidal directions. Since the potential equations then do not depend on the poloidal coordinate, it is efficient to divide the parallelization domain in this direction. The technique for solving Eqs. (5 and 6) is to perform fast fourier transforms (FFTs) in the toroidal direction and finite differences in the radial direction. Because these potential equations are linear, the solution for ϕ and $A_\parallel$ requires only a tridiagonal inversion in the radial direction and the FFT; both operations are localized to each poloidal domain.

To study realistic problems, BOUT obtains magnetic geometry data and plasma profiles from global data files written by UEDGE. The magnetic data come ultimately from a magnetohydrodynamic (MHD) equilibrium code, and the plasma background profiles can be from a UEDGE solution or an analytic fit to experimental data. On a parallel machine, a pointer variable is set so that each processor only reads a subset of the data needed for its domain. Similarly, each processor writes and reads its own dump file for the data in its domain that can be used later to restart or continue the problem. Presently, a restarted problem needs to use the same number of processors as the original problem. For postprocessing, another program collects the data from a set of the dumped data files generated by BOUT and generates a single file for the global solution.

4.2. *Results*

For BOUT simulations, we also choose parameters corresponding to the edge plasma of the DIII-D tokamak [17]. The computation mesh has 64 poloidal, 64 toroidal, and 40 radial points. The equilibrium plasma profiles are taken from hyperbolic tangent fits to the DIII-D experimental data (discharge # 89840) at the midplane for plasma density, n_{i0}; electron temperature, T_{e0}; ion temperature, T_{i0}; the electric field profile; and zero parallel velocity. The midplane temperature and density on the separatrix are $T_{e0} = 58$ eV, $T_{i0} = 50$ eV, and $n_{i0} = 1.7 \times 10^{19}$ m^{-3}.

We first compare two implicit methods of advancing the equations in time as discussed in Section 2.3. One is the Adams functional iteration (using only one iteration step) and the second is an inexact Newton BDF method utilizing matrix-free Krylov projections. For this problem, increasing the number of functional iteration steps for the Adams method beyond the one-step predictor corrector does not result in significantly better performance. The simulations are begun with a small fractional noise component ($\sim 10^{-5}$) that evolves into fully developed turbulence. The estimated local relative-error tolerance in PVODE for each of the cases is set to 10^{-4}. The resulting time-step history of the two methods is shown in Fig. 6. At the beginning, both methods show small time steps, but soon the Newton–Krylov method is able to expand its time step by a factor of 70 compared to the predictor–corrector Adams method for the same accuracy. In the nonlinear stage of the simulation where different wave modes are strongly coupled, the Newton method reduces its time step by about $1/2$ to satisfy the accuracy constraint. In fact, this simulation includes the large shear in the magnetic equilibrium near the X-point and we could not successfully integrate this case with a previous predictor–corrector method (a one-step iteration). Thus, using the Newton–Krylov method has become an essential part of our generalized BOUT simulations.

A more relevant picture of performance of the predictor–corrector and Newton methods is obtained by comparing the total computational work. These results are given in Table I for the linear stage of the simulation shown at early time in Fig. 6. The linear stage pertains to times where amplitudes of the fluctuating Fourier components of the variables are sufficiently small that their evolutions are not significantly influenced by nonlinear interactions. Here the number of right-hand side (RHS) evaluations represents the large majority of the computation time required (no preconditioner is used for BOUT),

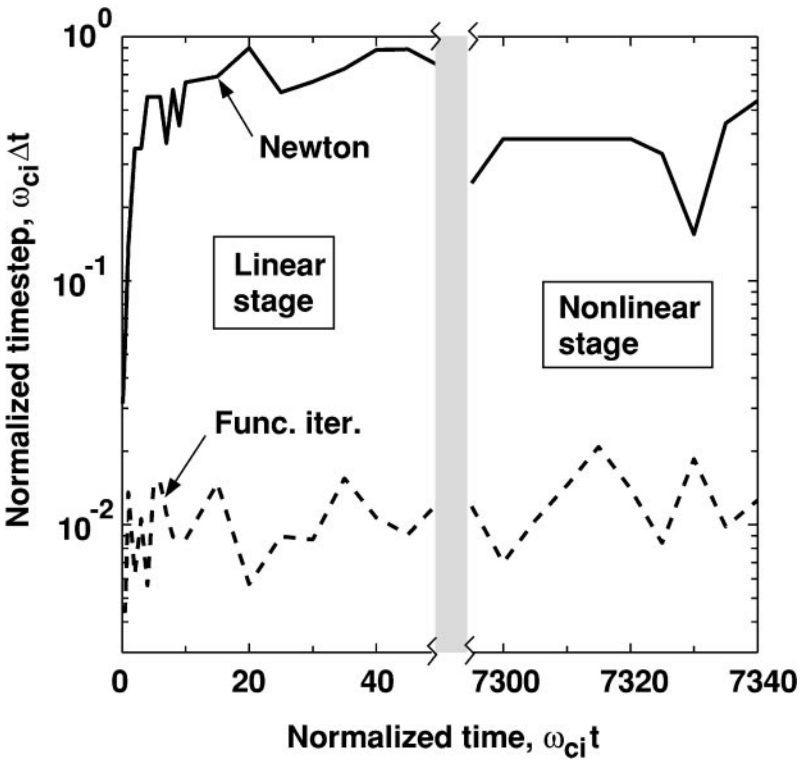


FIG. 6. Time step allowed in BOUT over the course of a time-dependent simulation showing the improvement obtained with new Krylov solver PVODE (or CVODE on serial computers) compared to the previously used functional iteration method.

and the ratio of the numbers in this first column thus gives an approximate measure of the relative speed of the methods; for this example, the Newton method is thus about six times more efficient in the linear regime. The average number of Newton iterations per time step is ∼1.5, and the number of inner Krylov iterations per Newton step is ∼6. The time step is measured in terms of the inverse ion-cyclotron frequency, $1/\omega_{ci} \approx 10^{-8}$ s, and the average value quoted is that after the very early transient where Δt increases rapidly. The fastest time scale in the problem is the parallel B-field length divided by the Alfven speed [$B(\mu_0 m_i n_i)^{-1/2}$ in SI units], giving 10^{-8} s for the parameters here. The order of the integration scheme is equal to the number of previous values of the RHS or of the variables used [see k in Eqs. (8 and 9)]. The value of k is chosen by PVODE for both methods to optimize performance, which results in $k = 1$ for the Adams method, but it is primarily the

TABLE I
Comparison of Adams Predictor–Corrector and Newton–Krylov (BDF) Statistics in Linear Stage of the Simulation

Method	Number of RHS evaluations	Number of time steps	Average $\Delta t\omega_{ci}$	Observed Δt order
One-step P/C	6212	5756	1×10^{-2}	1
BDF Newton	1091	115	7×10^{-1}	3–4

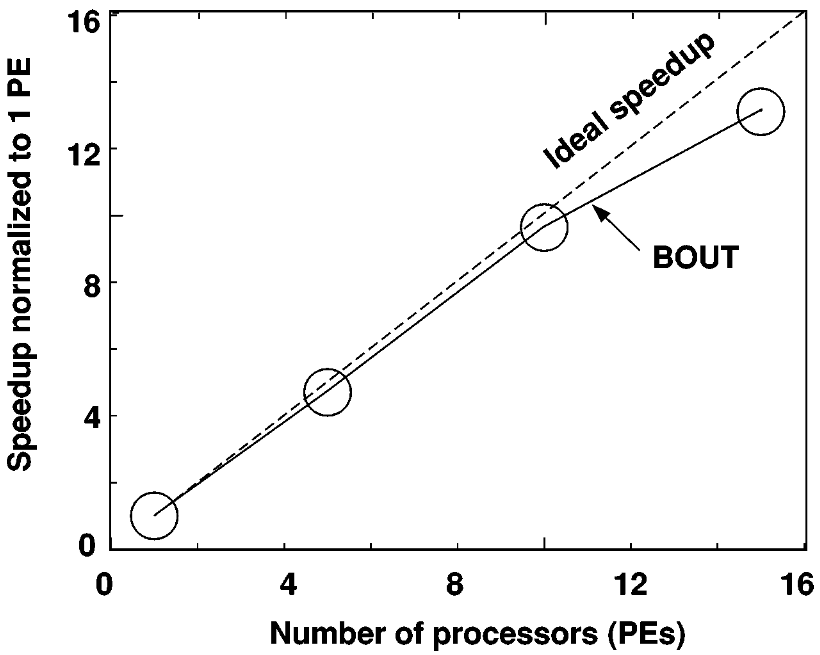


FIG. 7. Comparison of speed of parallel BOUT runs on the LLNL SUN Wildfire system with 16 processors; the runs used 1, 5, 10, and 15 processors. Only poloidal decomposition is used with no preconditioner.

Newton aspect of Newton–Krylov method that makes it superior. In the nonlinear regime, the Newton method is about three times more efficient than the Adams method (see Fig. 6).

To extend these improvements to parallel machines, we developed a parallel version of BOUT based on domain decomposition as described in Section 2.4. Because the potential equations (5 and 6) are independent of the poloidal dimension in the ballooning-coordinate representation, the most effective choice of domains are those that segment the poloidal direction. Thus, in referring to Fig. 3, this would consist of removing the horizontal dotted lines and combining domains (0, 4, 8, 12), (1, 5, 9, 13), etc. Using these poloidal domains, the solution of Eqs. (5 and 6) can be done entirely on each domain without regard to the other domains. Then, only message passing is required to fill the guard cells of each domain in order to use PVODE.

The effectiveness of the parallel BOUT code on a SUN Wildfire system is shown in Fig. 7. This parallel system has 16 processors per machine. Although this system has three machines, we only used one because of some initial intermachine scheduling problems. Here and elsewhere, the speedup time refers to wall-clock time, but we verified that it was close to the CPU time since we were the sole user of the 16-processor machine during these tests. Cases of 1, 5, 10, and 15 processors are shown in Fig. 7. The speedup is nearly linear, with a modest degradation at 15 processors. The source of the degradation was not investigated, but it is not fundamental to our problem as the next example shows.

When this problem was run on the T3E-900 at NERSC, we could more effectively study the code's behavior through 15 to 60 or more processors. The results are shown in Fig. 8. One can see that the speedup is actually super-linear over the range considered when normalized to the case using 5 processors, which is the smallest number of processors we could fit this problem into. The super-linear behavior, or offset linear at high processor number, is

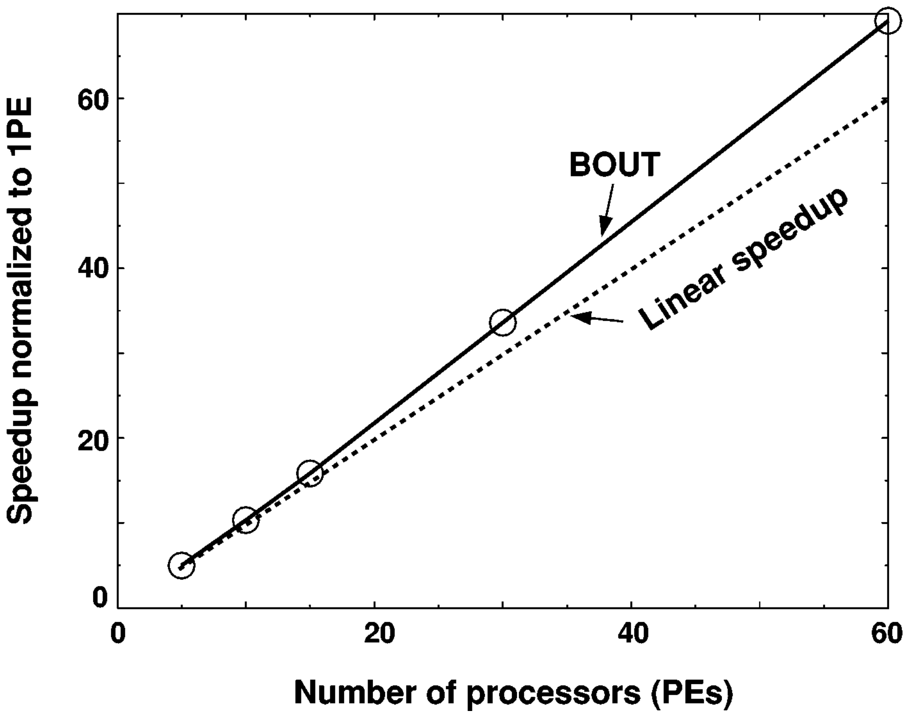


FIG. 8. Comparison of speed of BOUT runs with various numbers of processors on the NERSC CRAY T3E-900. Only poloidal decomposition is used with no preconditioner. The super-linear behavior, or offset linear curve, is likely caused by better utilization of fast cache memory for a large number of processors.

likely caused by the access speed and size of different types of CPU memory available on the T3E. For the 5-processor case, the memory required per processor is significantly larger than that available in the fast cache memory, while for the 60-processor case, a larger percentage of the calculation can reside in the fast cache memory. The division of work for the 60 processor case is 81% for evaluating the BOUT physics equations, 12% for internal PVODE calculations, 6% for interprocessor MPI communications, and 1% for other overhead costs. The load balance among processors is very good with only a ∼1% variation.

5. CONCLUSIONS

We have succeeded in developing parallel versions of two workhorse codes to simulate edge plasmas in MFE devices: UEDGE for 2-D transport and profile evolution and BOUT for 3-D turbulence. Both codes solve the magnetized plasma fluid equations, with UEDGE focusing on long-time evolution of the plasma profiles and BOUT dealing with short-time turbulence that causes anomalous radial transport. A similar domain-decomposition model is used to achieve the parallelization, which then allows us to utilize the recently developed Newton–Krylov solver PVODE [3].

The parallelization of UEDGE allowed us to obtain nearly an order of magnitude speedup in execution time for the plasma equations on 16 processors [16]. We were able to reuse almost all of the original FORTRAN coding. We developed a domain-decomposition model including an automatic decomposition routine and a number of message-passing routines, and we tested and debugged interface routines with the PVODE solver. We plan to extend this work to the time-independent parallel nonlinear solver KINSOL [3], which is the parallel equivalent of the serial NKSOL solver.

The fluid gas equations do not parallelize as effectively as the plasma equations because of the anisotropic mesh and lack of domain overlap in the preconditioner. We believe that providing more overlap information in the preconditioner may allow this problem to be overcome, such as using a Schur complement method [12, 20] or other other schemes [10]. Also, when using a Monte Carlo neutrals code for the gas description [21], this problem goes away, and one gets the added benefit that Monte Carlo codes parallelize very well. On the other hand, one must then achieve convergence of the separate plasma and neutral descriptions by an iteration procedure [22].

The results for the BOUT 3-D code exceeded our initial expectations. Even before parallelization, the conversion to the Newton–Krylov solver [3] produced a code that runs as much as six times faster than an Adams functional iteration method for the case studied and then decreases to three times faster in the strongly turbulent region. These simulations are very important for understanding the behavior of present experiments and designing future devices [18, 19].

The parallelized version of BOUT continues to work well with a poloidal domain decomposition, giving a factor of 13 speedup for 15 processors on the SUN Wildfire and a very encouraging factor of 69 speedup for 60 processors on the T3E-900. The modest degradation on the Wildfire system was not studied. The super-linear speedup on the T3E is likely due to the better utilization of cache memory for the larger number of processors. Most recently, we extended this case to 120 processors on the T3E and found the data on the same offset linear curve. Note that since BOUT presently uses no preconditioner, this speedup is due entirely to the parallelization; i.e., there is no contribution from possible modifications to the preconditioner from the domain decomposition.

There are two areas where more short-term improvements may be realized with BOUT performance. One is to extend the domain decomposition to the radial direction as in UEDGE. This will allow the use of more domains as the number of allowable toroidal modes increases. Here we will deal with the coupling of the potential equations across the radial domains by a parallel tridiagonal solver [23] or a Newton–Krylov solver using a preconditioner. The second area we are focusing on is to increase the time step of the PVODE integration by providing a preconditioner for the time-dependent equations. This gain has limitations in that we must still properly resolve the turbulence. Some simple preconditioners were tried without much improvement, but we know from our experience with UEDGE that preconditioners can be effective for the equation set we are using, and this warrants further investigation.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under Contract W-7405-ENG-48. This research was supported by the LLNL LDRD program as Project 97-ERD-045. We have benefited from many useful discussions with A. G. Taylor and P. N. Brown. The simulations on the SUN Wildfire and DEC computers were performed at LLNL while the simulations on the T3E-600 and T3E-900 computers were performed at NERSC. We thank the LLNL Computing Center staff and the NERSC staff for their able assistance; help from Chris H. Q. Ding of NERSC is especially acknowledged.

REFERENCES

1. T. D. Rognlien, J. L. Milovich, M. E. Rensink, and G. D. Porter, A fully implicit, time dependent 2-D fluid code for modeling tokamak edge plasmas, *J. Nucl. Mater*. **196–198**, 347 (1992); T. D. Rognlien, P. N. Brown, R. B. Campbell, T. B. Kaiser, D. A. Knoll, P. R. McHugh, G. D. Porter, M. E. Rensink, and G. R. Smith, 2-D fluid transport simulations of gaseous/radiative divertor, *Contrib. Plasma Phys.* **34**, 362 (1994).
2. X. Q. Xu and Ronald H. Cohen, SOL turbulence theory and simulations, *Contrib. Plasma Phys.* **38**, 158 (1998).
3. A. C. Hindmarsh and A. G. Taylor, *PVODE and KINSOL: Parallel Software for Differential and Nonlinear Systems*, Lawrence Livermore National Laboratory Report UCRL-ID-129739, (Feb. 1998).
4. P. N. Brown and A. C. Hindmarsh, Reduced storage matrix methods in stiff ODE systems, *J. Appl. Math. Comput.* **31**, 40 (1989).
5. P. N. Brown and Y. Saad, hybrid Krylov methods for nonlinear systems of equations, *SIAM J. Sci. Stat. Comput*. **11**, 450 (1990).
6. G. R. Smith, P. N. Brown, R. B. Campbell, D. A. Knoll, P. R. McHugh, M. E. Rensink, and T. D. Rognlien, Techniques and results of tokamak-edge simulation, *J. Nucl. Mater*. **220–222**, 1024 (1995).
7. D. A. Knoll and P. McHugh, NEWEDGE: A 2-D Fully Implicit Edge Plasma Fluid Code for Advanced Physics and Complex Geometry, *J. Nucl. Mater*. **196–198**, 352 (1992).
8. B. F. Smith, P. E. Bjorstad, and W. D. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations* (Cambridge Univ. Press, Cambridge, UK, 1996).
9. W. D. Gropp, E. Lusk, and A. Skjellum, *Using MPI Portable Parallel Programming with the Message-Passing Interface*, (MIT Press, Cambridge, MA, 1994).
10. D. A. Knoll, P. R. McHugh, and V. A. Mousseau, Newton-Krylov-Schwarz methods applied to the tokamak edge plasma fluid equations, in *Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering*, edited by D. E. Keyes, Y. Saad, and D. G. Truhlar (Soc. for Industr. & Appl. Math., Philadelphia, 1995), pp. 75–95.
11. Y. Saad, ILUT: A dual threshold incomplete ILU factorization, *Numer. Linear Algebra Appl.* **1**, 387 (1994).
12. Yousef Saad, *Iterative Methods for Sparse Linear Systems* (PWS Publishing Co., Boston, 1996).
13. S. I. Braginskii, Transport processes in a plasma, in *Reviews of Plasma Physics*, Vol. I, edited by M. A. Leontovich (Consultants Bureau, New York, 1965), p. 205.
14. J. A. Wesson, *Tokamaks*, 2nd ed. (Oxford Univ. Press, Oxford, UK, 1997).
15. C. William Gear, *Numerical Initial Value Problems in Ordinary Differential Equations* (Prentice-Hall, Englewood Cliffs, NJ, 1971).
16. T. D. Rognlien, X. Q. Xu, A. C. Hindmarsh, P. N. Brown, and A. G. Taylor, *Algorithms and Results for a Parallelized Fully-Implicit Edge Plasma Code*, Int. Conf. Numer. Sim. Plasmas, Feb. 10–12, 1998, Santa Barbara, CA; LLNL Report UCRL-JC-129223-abs.
17. J. L. Luxon, P. Anderson, F. Batty, *et al.*, Initial results from the DIII-D tokamak, in *Proc. 11th Int. Conf. Plasma Phys. Controlled Nucl. Fusion* (IAEA, Vienna, 1987), p. 159.
18. X. Q. Xu, R. H. Cohen, T. D. Rognlien, and J. R. Myra, Low-to-high confinement transition simulations in divertor geometry, *Phys. Plasmas* **7**, 1951 (2000).
19. X. Q. Xu, R. H. Cohen, G. D. Porter, T. D. Rognlien, D. D. Ryutov, J. R. Myra, D. A. D'Ippolito, R. Moyer, and R. J. Groebner, Turbulence studies in tokamak boundary plasmas with realistic divertor geometry, *Nucl. Fusion* **40**, 731 (2000).
20. E. T. Chow, private communication (1998).
21. M. E. Rensink, L. LoDestro, G. D. Porter, T. D. Rognlien, and D. P. Coster, A comparison of neutral gas models for divertor plasmas, *Contrib. Plasma Phys*. **38**, 325 (1998).
22. D. Reiter, Progress in two-dimensional edge plasma modeling, *J. Nucl. Mater*. **196–198**, 80 (1992).
23. N. Mattor, T. J. Williams, and D. W. Hewett, Algorithm for solving tridiagonal matrix problems in parallel, *Parallel Comput.* **21**, 1769 (1995).