UCRL-JC-138518
PREPRINT

Application of Parallel Implicit Methods to Edge-Plasma Numerical Simulations

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

This paper was prepared for submittal to
Physical Review E

April 12, 2000

Lawrence Livermore National Laboratory

Application of Parallel Implicit Methods to Edge-Plasma Numerical Simulations

T.D. Rognlien, X.Q. Xu, and A.C. Hindmarsh
Lawrence Livermore National Laboratory
Livermore, CA 94551

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, 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 two recently-developed Newton-Krylov numerical solvers, PVODE and KINSOL. Results show an order of magnitude speed up in execution time for the plasma equations with UEDGE. 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 is closer to two orders of magnitude, especially when one includes the initial improvement from switching to the fully implicit Newton-Krylov solver. 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.

Contents

I. Introduction

The goal of this work is to develop a 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 maintaining 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. The last several years 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 multi-processor computers. For codes with explicit time advancement, this shift is relatively straightforward because only local 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 which render the equations describing them "stiff" in the numerical sense. Here, implicit methods are especially useful, but implicit methods are inherently nonlocal spatially 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: UEDGE[1] solves for the slowly-evolving two-dimensional (2-D) profiles of a multi-species 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. 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 greatly from the development of two 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. 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, for such solvers to work well for UEDGE, our experience on serial computers shows that the system of equations must be well preconditioned. The preconditioning step is to solve a closely related problem in a more efficient but approximate manner which, in effect, serves as a guess for the final solution.[4,5] Thus, part of our work for UEDGE focuses on development of a parallel preconditioner based on a domain decomposition model. This Fortran preconditioner is interfaced to the C-language solvers PVODE and KINSOL[3] on a variety of parallel computer platforms 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 MPI package.[6] 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. Also, the serial version of UEDGE has been made very productive through the use the BASIS system,[9] but this programming and computing environment is not available on all computer platforms. This problem has been alleviated by recent development by Grote[11] of a conversion capability from BASIS to the more widely available PYTHON system,[10] and application of that conversion to UEDGE by Yang.[12]

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 functional iteration. Second, we find that the implicit BOUT works well without a preconditioner. 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 takes 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 report is as follows: In Sec. II, the geometry and basic equations for the edge-plasma problem are given. The domain decomposition model used for UEDGE and BOUT is described in Sec. IID. The results for the UEDGE parallelization are shown in Sec. III. The BOUT results from the fully implicit solver and parallelization are given in Sec. IV. The conclusions are summarized in Sec. V.

II. Equations, Geometry, and Algorithms

A. Basic equations

The basic models in the UEDGE and BOUT codes are obtained from the plasma fluid equations of continuity, momentum, and thermal energy for both the electrons and ions as 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

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

Here $m_{e,i}$ 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, $\mathbf{F}_{e,i} = \nabla \cdot \mathbf{\Pi}_{e,i}$ are the viscous forces, and $\mathbf{R}_{e,i}$ are the thermal forces.[13] The source $\mathbf{S}^m_{i,e}$ contains a sink term $-n m_{i,e} \mathbf{v}_i S^p_{i,e}$ which arises if newly created particles have no drift motion.

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

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

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

In their general three-dimensional form, the six equations given above represent ten 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. To calculate 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, $A_{||}$. Also UEDGE assumes symmetry in a third dimension, usually the toroidal direction for a fusion device, whereas BOUT allows fluctuations to have variations in all 3 spatial dimensions even though the equilibrium profiles are toroidally symmetric.

B. 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 fusion devices, we have been most interested recently in 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.

UEDGE and BOUT use the the poloidal flux surfaces as one spatial coordinate. The second spatial coordinate is often the curves normal to the flux surfaces also shown in Fig. 1a, but a nonorthogonal mesh is sometimes used to form the mesh along material surfaces at the boundary. For UEDGE, this specifies the 2-D coordinate system and all quantities are assumed uniform in the third direction. However, BOUT allows fluctuations to arise in the third toroidal direction, even though the equilibrium is uniform. Here, a segment of the torus is simulated as shown in Fig. 1b, which is then assumed to be periodically replicated to fill out the full 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.

C. 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

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u}) \tag{4}$$

where $\mathbf{u}$ is the vector of unknowns, and $\mathbf{f}$ is the result of sources, sinks, and the discretized spatial transport. 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 $\Delta t \rightarrow \infty$. 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 the later results sections (Sec. III and Sec. IV).

We consider two implicit schemes, one utilizing functional iteration and the other using the Newton-Krylov approach. Our codes presently use the Newton-Krylov method, but introducing the functional iteration method allows us to make a comparison later in the paper. These schemes are exemplified by the the Adams method and the Backward Differentiation Formula (BDF) method, respectively.[15] For the Adams method, the advancement of $\mathbf{u}$ from time level $n-1$ to n takes the form

$$\mathbf{u}_n = \mathbf{u}_{n-1} + \Delta t(\alpha_0 \mathbf{f}_n + \ldots + \alpha_{k-1}\mathbf{f}_{n-k+1}) \tag{5}$$

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

$$\mathbf{u}_n = (\beta_1 \mathbf{u}_{n-1} + \ldots + \beta_k \mathbf{u}_{n-k}) + \Delta t \gamma_0 \mathbf{f}_n \tag{6}$$

where the β's and γ_0 are coefficients determined by the order (k) used. 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 non-stiff systems. While the BDF method can also use functional iteration, a more effective method is often a Newton iteration which expands $\mathbf{f}_n$ at iteration j as

$$\mathbf{f}(\mathbf{u}^j) \approx \mathbf{f}(\mathbf{u}^{j-1}) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{u}^j - \mathbf{u}^{j-1}). \tag{7}$$

Equation (6) then is a linear equation for $\mathbf{u}_n^j$ which can be written as

$$(\mathbf{I}/\Delta t \gamma_0 - \mathbf{J})\mathbf{u}_n^j = \mathbf{g} \tag{8}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{J} \equiv \partial \mathbf{f}/\partial \mathbf{u}$ is the Jacobian evaluated with $\mathbf{u}$ from a previous iterate or time step. Also, $\mathbf{g}$ is a vector which depends on values of $\mathbf{u}$ from the past iteration, $\mathbf{u}^{j-1}$, and at previous time steps as obtained from Eqs. (6-7). Equation (8) 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.[4,7] Although more work is 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 code example. Also, UEDGE requires the inexact Newton method for reasonable performance, and it also requires that the system of equations be preconditioned.

Newton schemes that utilize a matrix-free Krylov projection method often require preconditioning.[4,7] The procedure requires the ability to solve related linear systems $\mathbf{Pw} = \mathbf{h}$ with a matrix $\mathbf{P}$ which 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. (8) to form the preconditioned system

$$[(\mathbf{I}/\Delta t\gamma_0 - \mathbf{J})\mathbf{P}^{-1}](\mathbf{P}\mathbf{u}_n^j) = \mathbf{g}. \tag{9}$$

The new variables are $\mathbf{Pu}_n$, and this system is easier to solve by iterative methods such as the Krylov method since $(\mathbf{I}/\Delta t\gamma_0 - \mathbf{J})\mathbf{P}^{-1} \equiv \mathbf{A} \sim \mathbf{I}$ is more diagonally dominant. The Krylov method does require matrix-vector products of $\mathbf{A}(\mathbf{Pu}_n)$, and these are done in a matrix-free manner with a finite-difference quotient approximation to $\mathbf{Jv}$.[4]

D. Domain Decomposition Model

Because UEDGE and BOUT use the same poloidal mesh, this region can be divided into domains on parallel computers where separate processors can solve a local problem. However, for the toroidal 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 poloidal geometry 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.

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 sixteen 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[6] in order to fill the guard cells that surround each domain shown in Fig. 3b. Notice that data needed in a guard cell is not necessarily from the adjacent domain; *e.g.*, the right-side guard-cell data for domain 0 in Fig. 3a comes from domain 3. These guard cells do not contain variables that are advanced for each domain, but rather only contain 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 boundaries conditions are local, so no message-passing between domains is required.

The domain decomposition plays two roles. First, in order to utilize the implicit PVODE and KINSOL solvers,[3] we must divide the physical space of our codes in this manner, where each processor has all of the information required to solve the local problem. Because these Newton-Krylov implicit solvers use a local, finite-difference approximation to the Jacobian, there is no need to invert a global matrix. The second use of the domain decomposition model is that it provides the basis for the preconditioning algorithm which is required by UEDGE. Here the full set of Jacobian elements can be efficiently generated in parallel for each domain using all of the processors. However, to obtain a preconditioner, one needs to approximate the inversion of the Jacobian which is more nonlocal than the Krylov approximation. To do this, we perform an approximate LU decomposition of the Jacobian using ILUT[7] routines independently on each processor. This procedure of not including coupling between domains at the preconditioning level is referred to as the additive-Schwartz method.[8]

The manner in which UEDGE utilizes the solvers PVODE and KINSOL 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. In addition, UEDGE provides a local preconditioning Jacobian on each processor by approximating derivatives with finite-difference quotients. This operation should not be confused with the finite-difference approximation used by the PVODE and KINSOL Krylov solvers. In the central column of the diagram are the wrappers mentioned previously which 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 have been developed previously.[3] Note that the foregoing model is replicated for all domains or processors. Communication between processors as required to fill that 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 here.

III. Implementation and Results for UEDGE

A. Implementation

The 2-D plasma transport equations used in UEDGE come from a reduction of those presented in Sec. IIA for the parameters of the edge plasma. This results in five equations for the following variables: ion density, n_i; ion parallel velocity, $v_{||}$; separate electron and ion temperatures, T_e and T_i; and the electrostatic potential, ϕ. In addition, impurity species can be included which have their own density and parallel velocities but a common temperature, T_i, with the ions. Furthermore, neutral gas species are present, which can be described by various models; the simplest is a diffusion model, where the neutral density, n_n, obeys the continuity equation

$$\frac{\partial}{\partial t}n_n + \frac{\partial}{\partial x}(n_n v_{nx}) + \frac{\partial}{\partial y}(n_n v_{ny}) = (\langle\sigma_r v_e\rangle - \langle\sigma_i v_e\rangle)n_e n_n. \tag{10}$$

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 the results section.

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. The most effective preconditioner is forming the full Jacobian matrix by finite-difference quotients as mentioned previously. We now have two options for the parallel UEDGE, either using an algorithm developed within UEDGE, or because each domain with guard cells is now self-contained, we can use the band-block-diagonal preconditioners PVBBDPRE and KINBBDPRE supplied as part of the PVODE and KINSOL packages, respectively.[3]

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 sub-domains in each of these regions, and the algorithm works to optimize the load balancing by having any imbalance in the number of equations per domain being relegated to a minimum number of processors which do less work than the average. In this way, most processors do not need to wait for the unbalanced ones with fewer equations to finish. Usually, we strive for complete load balancing by a proper choice of mesh sizes and number of domains. The newly written 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. This data includes 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.

B. Results

We have 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. The computation mesh has 64 poloidal mesh points and 48 radial points. 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 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 computer time required to find the solution when we switch to $T_{e,i} = 150$ eV on the core boundary. The execution time normalized to that for one processor is presented in Fig. 5. Here PVODE was used to run to steady state with the plasma equations, and two different preconditioners were used, the case marked X being PVBBDPRE, and the + point being the internal UEDGE preconditioner. In the table below the figure, the tabulated data shows the number of function (or right-hand side) evaluations, the number of preconditioners, 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 are working 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 X) has only about 1/3 of the preconditioner evaluations compared to the + data point with the UEDGE preconditioner. As a consequence, the X data point has almost twice the number of overall function evaluations from PVODE. Thus, the results from the two preconditioners indicate the sensitivity of the trade-off between more frequent preconditioner evaluations (and LU decompositions), and fewer Newton-Krylov iterations as reflected in the function evaluation count. The extra work required because the preconditioner is only solved locally on each domain and thus does not include domain-coupling information is reflected in the lower number of function and preconditioner evaluations for the 1 processor base-case.

While it is encouraging to obtain nearly an order of magnitude speed up for the plasma equations in UEDGE, the relatively simple gas equation shown by Eq. (10) proved more difficult. This problem has been traced back to the highly anisotropic mesh shown in Fig. 1.[16] This mesh is chosen to best represent the plasma which flows rapidly along the long 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 find the same difficulty. We believe that providing more overlap information in the preconditioner should allow this problem to be overcome, such as using a Schur complement method.[8,17] Also, when coupling to a Monte Carlo neutrals code for the gas description,[18] this issue 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.[19]

IV. Implementation and Results for BOUT

A. Implementation

For edge-plasma turbulence, the application of a fluid model is reasonable in part because of the low temperature and high collisionality. While the unstable modes can have wavelengths short compared to the scale lengths of equilibrium profiles, the dominant modes have perpendicular wavelengths which are larger than the ion gyroradius, ρ_s, consistent with a fluid approach. Thus, it is again appropriate to use the Braginskii fluid equations as presented in Sec. IIA. By scaling arguments, the full set of fluid equations can be reduced to a seven-variable set for the electrostatic potential, ϕ; magnetic vector potential, $A_{||}$; plasma density, n_i; electron and ion temperatures, T_e and T_i; and electron and ion parallel velocities $v_{e,i||}$. The parallel current, $j_{||}$, and vorticity, ϖ, are intermediate variables used to help solve the system. The equations for all of the variables satisfy time-evolutionary equations, except for ϕ and $A_{||}$. These potentials satisfy similar equations:

$$\nabla_\perp^2 \phi = \varpi \tag{11}$$

$$\nabla_\perp^2 A_{||} = -\frac{4\pi}{c} j_{||}. \tag{12}$$

The ϕ potential equation is not obtained from Poisson's equation, but rather from the quasineutrality condition and the current continuity equation. Here $\nabla_\perp^2$ refers to the Laplacian operator in the directions perpendicular to the magnetic field. It is the solution to these rather simple looking equations that has important consequences for the parallel version of BOUT.

In order to efficiently simulate turbulence with short perpendicular wavelengths compared to parallel wavelengths (*i.e.*, for wavenumbers $k_{||} \ll k_\perp$), 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 $\nu \equiv a_e B_t / R B_p$, 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: $\partial/\partial\psi = \partial/\partial x - (\int \partial\nu/\partial\psi)\partial/\partial z$, $\partial/\partial\theta = \partial/\partial y - \nu\partial/\partial z$, $\partial/\partial\varphi = \partial/\partial z$, and $\nabla_{||} = (B_p/a_e B)\partial/\partial y$. 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. (11-12) 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 with short toroidal wavelengths reduces the potential equations to two dimensions in the radial and toroidal directions by considering modes where $\partial/\partial\theta \approx -\nu\partial/\partial z$ as discussed in the previous paragraph. 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. (11-12) is to perform Fast-Fourier Transforms (FFT) in toroidal direction and finite differences in the radial direction. Because these potential equations are linear, the solution for ϕ and $A_{||}$ only requires 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 comes ultimately from an 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 which later can be used to restart or continue the problem. Presently, a restarted problem needs to use the same number of processors as the original problem. For post-processing, 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.

B. Results

For the BOUT simulations, we also choose parameters corresponding to the edge-plasma of the DIII-D tokamak.[20] The system has the following basic parameters. 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 Sec. IIC: one is the Adams functional iteration and the second is the inexact Newton method utilizing matrix-free Krylov projections. The simulations are begun with a small fractional noise component ($\sim 10^{-5}$) which evolves into fully developed turbulence. The estimated local relative-error tolerance 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 functional-iteration 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 shear in the magnetic equilibrium near the X-point which was a problem that we could not integrate successfully with the previous predictor-corrector method (a one-step functional iteration). Thus, using the Newton-Krylov method has become an essential part of our generalized BOUT simulations.

The complete picture requires that the total computational work for the two methods be considered. These results are given in Table 1 for the linear stage of the simulation shown in Fig. 6. Here the number of right-hand side (RHS) evaluations represents the large majority

Method	# RHS eval	# time steps	Ave. time step	Δt order
Func. iter.	6212	5756	1×10^{-2}	1
Newton	1091	115	7×10^{-1}	3-4

Table 1: Comparison of functional iteration (Adams) and Newton-Krylov parameters in linear stage

of the computation time required, and the ratio of the numbers in this first column thus gives a approximate measure of the relative speed of the methods; for this example, the Newton method is thus ~ 6 times more efficient. The time step is measured in terms of the inverse ion-cyclotron frequency, $1/\omega_{ci}$, and the average value quoted is that after the very early transient where Δt increases rapidly. 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. (5-6)]. The value of k for the Newton method is chosen by PVODE to optimize performance, whereas we prescribe $k = 1$ for the functional iteration method to mimic typical integration methods. We find that allowing $k > 1$ for the functional iteration method does not significantly change the performance for our cases.

In order to extend these improvements to parallel machines, we developed a parallel version of BOUT based on domain decomposition as described in Sec. IID. Because the potential equations, Eq. (11-12), are independent of the poloidal dimension in the ballooning-coordinate representation, the most effective choice of domains are those which 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. (11-12) 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, 3 machines, and shared memory. Here and elsewhere, the speedup time refers to wall-clock time, but this is typically close to the CPU time. Note that the speedup is nearly linear on one machine, with a small degradation at 15 processors (15 rather than 16 because of load balancing for this given problem). However, when going to 30 processors, the speedup drops dramatically. This is caused by either slow message passing between machines compared to that between processors of one machine, or non-optimization of scheduling, issues which are being investigated. Also, we have used the MPICH implementation of MPI, whereas the use of SUN MPI may give improved performance on this SUN system. Nevertheless, the speedup with 15 processors is a factor of 13, which is very useful. Note that a data point is also shown for the DEC cluster at LLNL for 1 processor. BOUT is somewhat faster on the DEC cluster than the

SUN cluster for our problem using 1 processor. But more significantly, it is quite slow in the parallel mode because of scheduling issues, which prevent us from simultaneously obtaining all the processors (10) for even one machine in the 8-machine cluster. The scheduling issues are likely to be resolved and do not represent a fundamental limitation.

When this same problem is run on the T3E-900 at NERSC, one can more effectively study the behavior from 15 to 60 or more processors, and the results, shown in Fig. 8, are even more impressive. 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 off-set linear at high processor number, is likely caused by the speed of access to different types of CPU memory available on the T3E. For the 5 PE 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 between processors is very good with only a ~1% variation.

V. 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 which causes anomalous radial transport. A similar domain decomposition model is used to achieve the parallelization where we then utilize the recently developed LLNL Newton-Krylov solvers PVODE and KINSOL.[3]

The parallelization of UEDGE has allowed us to obtain nearly an order of magnitude speedup in execution time for the plasma equations on 16 processors.[16] Here we were able to reuse almost all of the original FORTRAN coding, although we did have to create a BASIS-free version of the code that could run on the T3E; with the automated conversion[11,12] to

PYTHON, this should not be needed in the future. We developed a domain decomposition model including an automatic decomposition routine and a number of message passing routines, plus tested and debugged interface routines with the PVODE and KINSOL solvers. 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. There are overlap methods which should be assessed for this problem. Also, the coupling of the parallel plasma equations with a parallel neutral Monte Carlo code looks promising.

The results for the BOUT 3-D code have exceeded our initial expectations. The conversion to the Newton-Krylov solver[3] has produced a code which runs as much as 50 times faster compared to an Adams functional iteration method. In fact, the previous predictor-corrector method we used, which is even simpler than the Adams functional iteration method, could not be practically used for the simulations which include X-point shear from the equilibrium magnetic field. These simulations are very important for understanding the behavior of present experiments and designing future devices.[21,22]

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 degradation on the Wildfire system at 15 processors may be due to inefficient message passing resulting from the use of MPICH instead of SUN MPI. 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 have extended this case to 120 processor on the T3E, and find the data on the same off-set linear curve. The large speedup of BOUT gives real optimism concerning coupling UEDGE and BOUT. Previously, BOUT was very time consuming, which made such coupling seem far off; now it is a real possibility.

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 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 being focused on is to increase the time step of the PVODE integration further by providing a preconditioner for the time dependent equations. This gain does have 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 equations set we are using, and this warrants further investigation.

Acknowledgments: This work was supported by the LDRD program at LLNL during Fiscal Years 1997-98 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 Livermore Computing Center staff and the NERSC staff for their able assistance; help from Chris H.Q. Ding of NERSC is especially noteworthy. This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

[1]T.D. Rognlien, P.N. Brown, R.B. Campbell, *et al.*, Contrib. Plasma Phys. **34**, 362 (1994).

[2]Xueqiao Xu and Ronald H. Cohen, 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, J. Appl. Math. Comp. **31**, 40 (1989).

[5]P.N. Brown and Y. Saad, SIAM J. Sci. Stat. Comput. **11**, 450 (1990).

[6]W.D. Gropp, E. Lusk, and A. Skjellum, *Using MPI Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, Cambridge, MA, 1994.

[7]Y. Saad, Num. Lin. Alg. Applic. **1**, 387 (1994).

[8]Yousef Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Co., Boston, MA, 1996.

[9]P.F. Dubois, *et al.*, "The Basis System", Lawrence Livermore National Laboratory Report UCRL-MA-118543, parts 1-6, 1994.

[10]Mark Lutz, *Programming Python* (O'Reilly & Associates, Sebastopol, CA 1996).

[11]D.P. Grote, private communication, 1998.

[12]T-Y.B. Yang, private communication, 1998.

[13]S.I. Braginskii, "Transport Processes in a Plasma", in *Reviews of Plasma Physics*, Vol. I, Ed. M.A. Leontovich (Consultants Bureau, New York, 1965), p. 205.

[14]J.A. Wesson, *Tokamaks*, 2nd Edition (Oxford University Press, Oxford, U.K., 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. Num. Sim. Plasmas, Feb. 10-12, 1998, Santa Barbara, CA.; LLNL Report UCRL-JC-129223-abs.

[17]E.T. Chow, private communication, 1998.

[18]M.E. Rensink, L. LoDestro, G.D. Porter, T.D. Rognlien, and D.P. Coster, Contrib. Plasma Phys. **38**, 325 (1998).

[19]D. Reiter, J. Nucl. Mater. **196-198**, 80 (1992).

[20]J.L. Luxon, P. Anderson, F. Batty, *et al.*, in Proc. 11th Int. Conf. Plasma Phys. Controlled Nucl. Fusion (IAEA, Vienna, 1987), p. 159.

[21]X.Q. Xu, R.H. Cohen, T.D. Rognlien, and J.R.Myra, Phys. Plasmas **7**, no.5 (2000).

[22]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, Nucl. Fusion **40**, 731 (2000).

[23]N. Mattor, T.J. Williams, and D.W. Hewett, Parallel Computing **21**, 1769 (1995).

Figures

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 which 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 also allows fluctuations to have toroidal variations which 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 increase computational cost.

FIG. 2. The poloidal plane is divided into 4 main regions for the domain decomposition model, each of which can be further subdivided. The 4 regions are mapped into the rectangular geometry shown in the lower part of the figure by opening the poloidal configuration along the dotted line.

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

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

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 X 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.

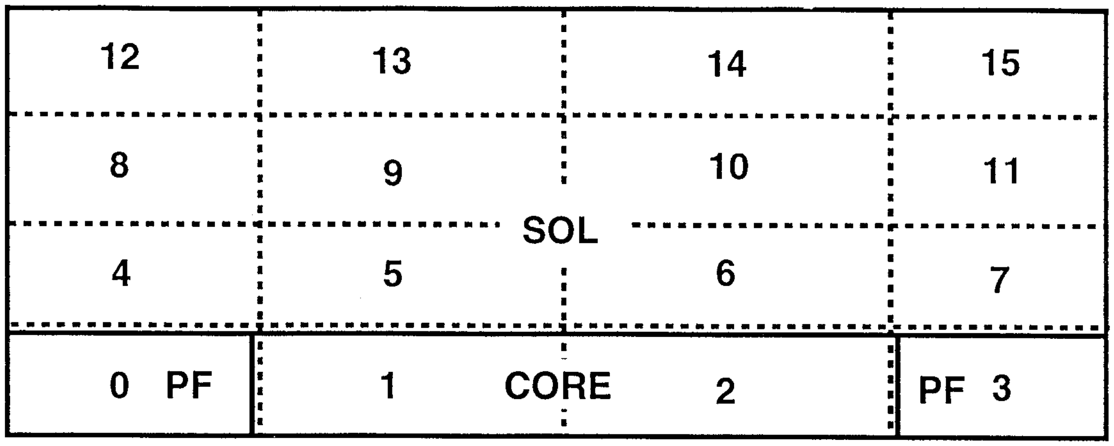
FIG. 6. Time step allowed in BOUT over the course of a time dependent simulation showing the improvement obtained with new Krylov solver PVODE (or CVODE on serial computers) compared to the previously-used functional iteration method.

FIG. 7. Comparison of speed of parallel BOUT runs on the LLNL SUN Wildfire system with 16 processors per machine. Only poloidal decomposition is used with no preconditioner. The drop from 15 to 30 processors is due to communication needed between

two computers; this may be caused by slow message passing or non-optimization of scheduling.

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 off-set linear curve, is likely caused by better utilization of fast cache memory for a large number of processors.

a)


12
13
14
15
8
9
10
11
SOL
4
5
6
7
0 PF
1
CORE
2
PF 3

b)

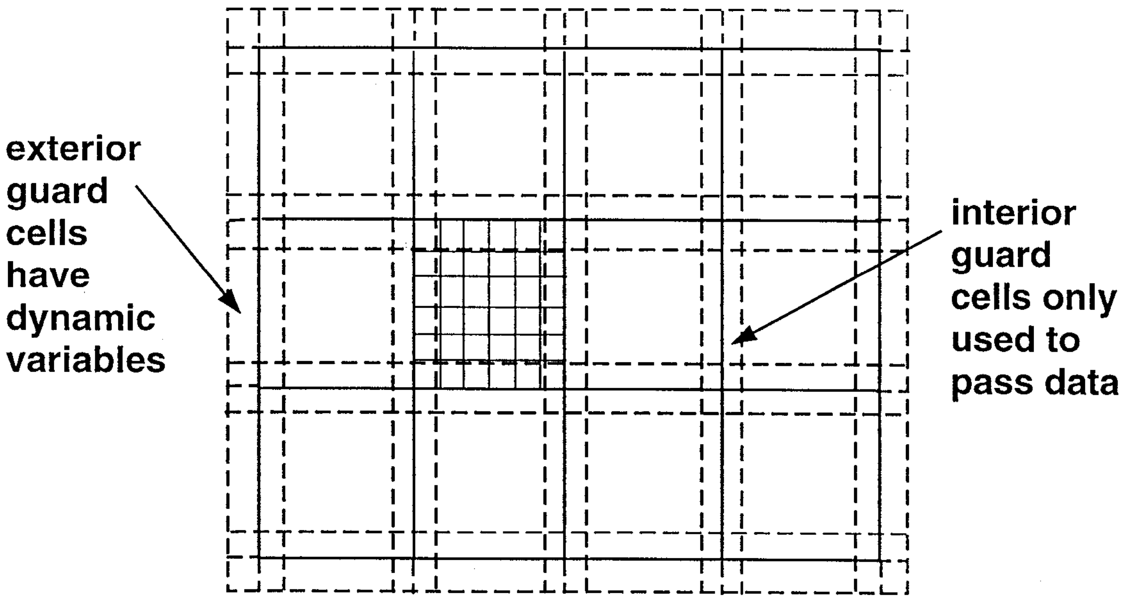

exterior guard cells have dynamic variables
interior guard cells only used to pass data

Fig. 3

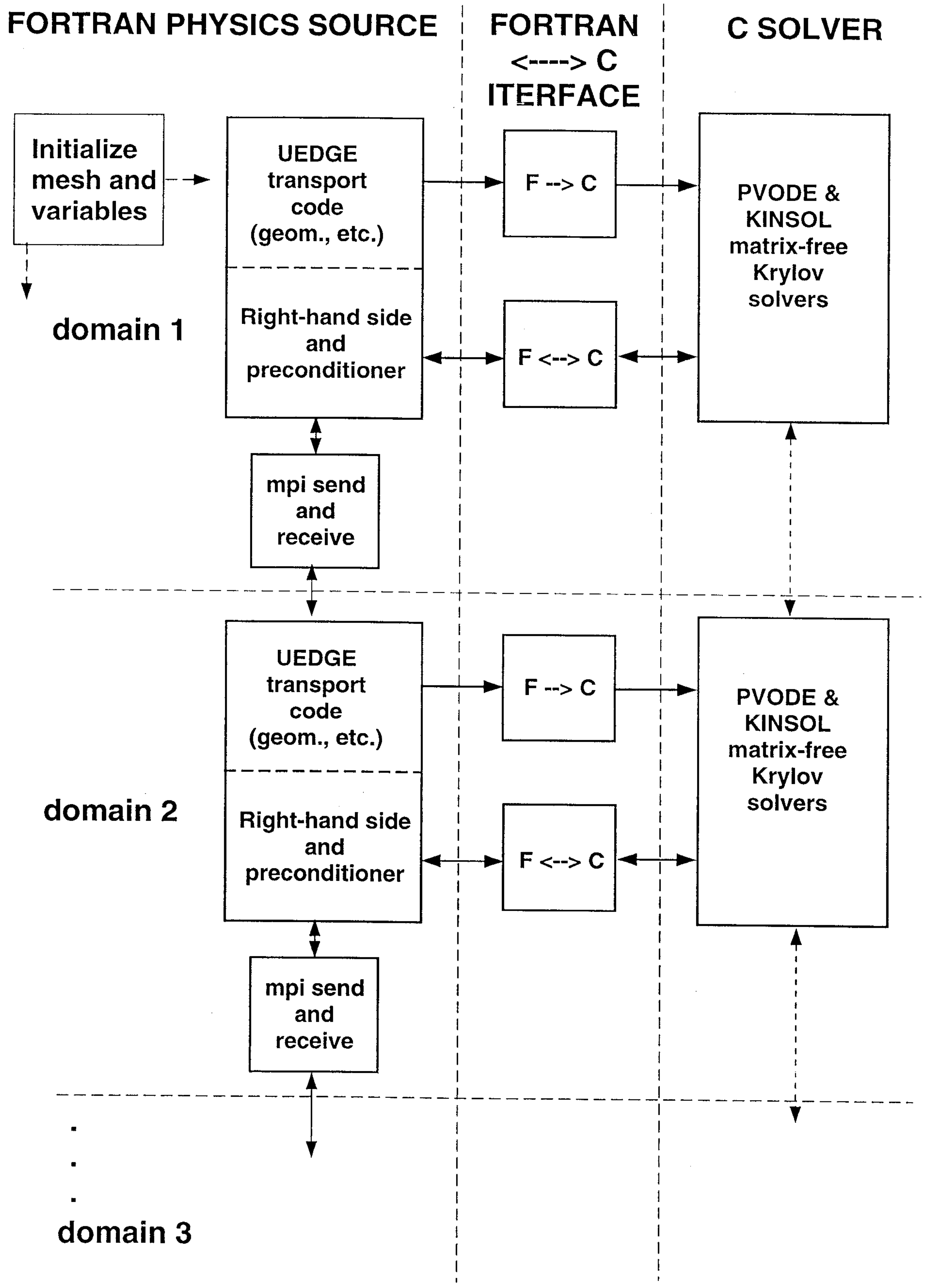
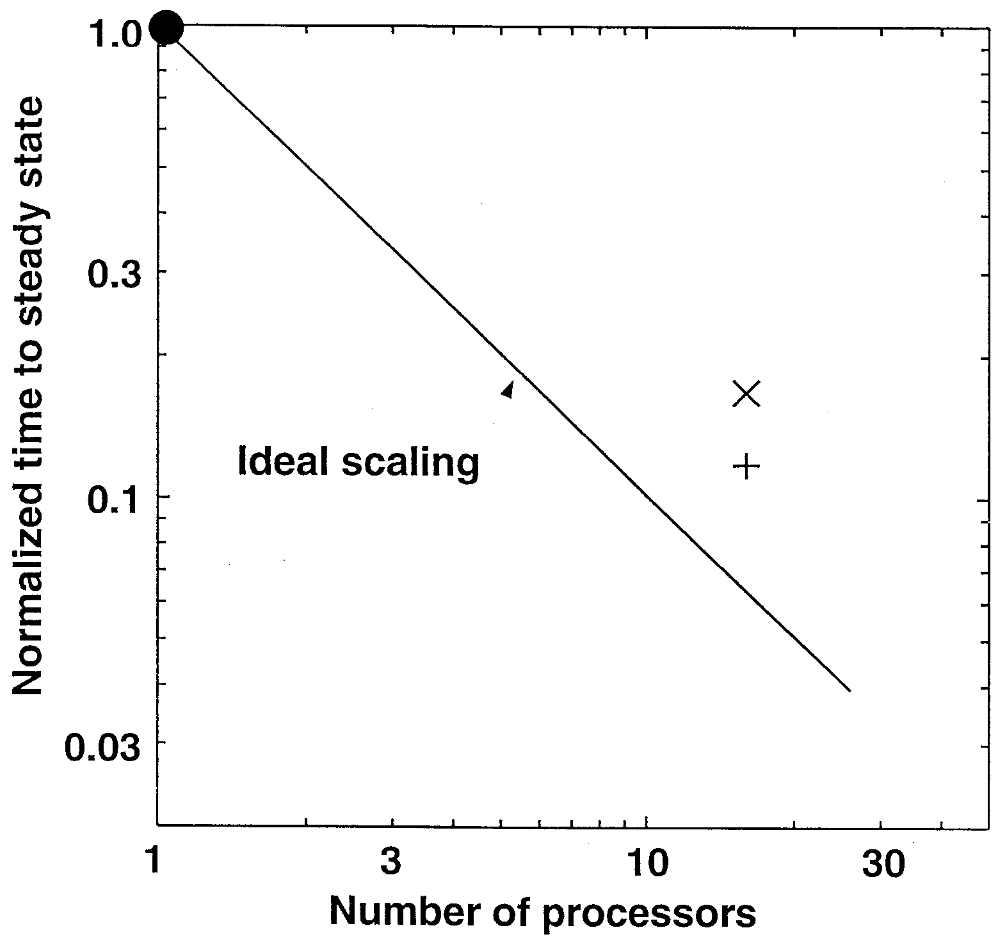
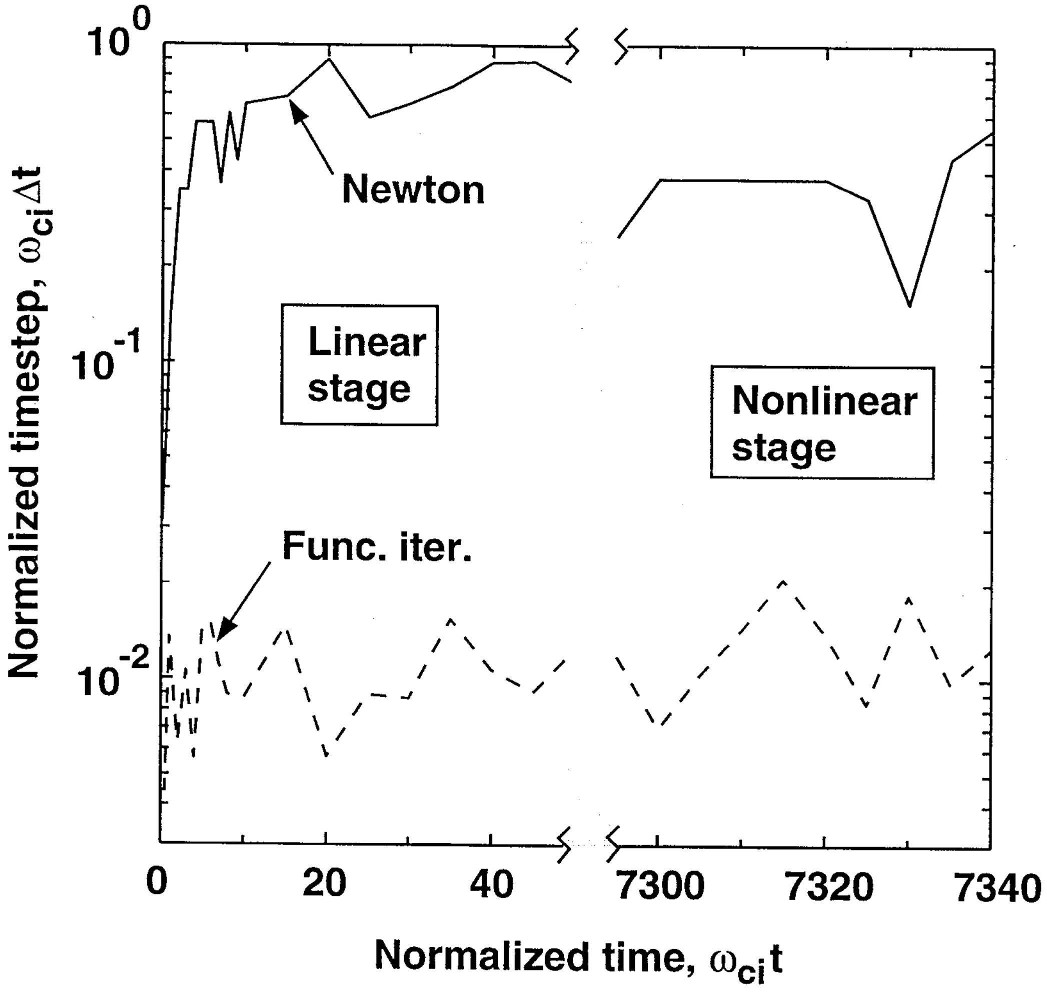

FORTRAN PHYSICS SOURCE
FORTRAN <----> C ITERFACE
C SOLVER
Initialize mesh and variables
UEDGE transport code (geom., etc.)
Right-hand side and preconditioner
F --> C
F <--> C
PVODE & KINSOL matrix-free Krylov solvers
mpi send and receive
domain 1
UEDGE transport code (geom., etc.)
Right-hand side and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

Fig. 4



Num. PEs	Func. eval	Prec. eval	Nrm. time	Ideal time
1 (●)	2236	9	1.0	1.0
16 (x)	9803	24	0.168	0.0625
16 (+)	5760	67	0.118	0.0625

Fig. 5



10^0
10^{-1}
10^{-2}
Normalized timestep, $\omega_{ci}\Delta t$
Newton
Linear stage
Nonlinear stage
Func. iter.
0
20
40
7300
7320
7340
Normalized time, $\omega_{ci}t$

Fig.6