Arbitrary Topology Equation Reader

D. A. Baver and J. R. Myra

Lodestar Research Corporation,
Boulder, Colorado, 80301

M. V. Umansky

Lawrence Livermore National Laboratory,
Livermore CA 94550

October, 2012

DOE/SC0006562 -1 LRC-12-148

Lodestar Research Corporation

2400 Central Avenue #P-5

Boulder, CO 80301

Arbitrary Topology Equation Reader
LRC Report Number 12-148 (October 2012)

D. A. Baver and J. R. Myra

Lodestar Research Corp., 2400 Central Ave. P-5, Boulder Colorado 80301

M. V. Umansky

Lawrence Livermore National Laboratory, Livermore CA 94550

Abstract

The Arbitrary Topology Equation Reader (ArbiTER) is a new code designed to solve linear partial differential equations in a wide variety of geometries. This can be used as a tool to analyze eigenmode structure and predict quantitative trends in growth rates. Moreover, by measuring the emergence of modes from a known equilibrium configuration, it can be used to quantitatively benchmark large-scale turbulence codes to a common standard. In addition to its utility as an eigenmode solver, the ArbiTER code is enormously flexible, with the ability to change physics models without modifying the underlying source code. A particularly new feature of this code is its ability to additionally vary the topology and dimensionality of the computational domain. This permits use of kinetic models, as well as the ability to study complicated geometries. The general principles of this code, as well as several benchmark cases, are presented here.

I Introduction

In this report, we describe a new code for solving eigenvalue problems in highly diverse geometries. This code is called the Arbitrary Topology Equation Reader, or ArbiTER. The name of this code refers to is relatively unique capability for rapid customization of model equations and computational domains. Since model equations are read from a file rather than incorporated into the source code, the program is termed an Equation Reader. Since topological information, such as connectivity and number of dimensions, are also read from an input file, the program is termed Arbitrary Topology. In practice, topology is not truly arbitrary, although its topological capabilities are remarkably diverse, and future upgrades to the code may bring it closer to this ideal.

The primary purpose of this code is to solve eigenvalue problems relevant to edge plasma turbulence. The secondary purpose of this code is to diversify into an undetermined number of other problems, applying the investment in code infrastructure to any problem for which the flexibility of the code makes it convenient to do so.

As an eigenvalue solver, its relevance to edge turbulence is primarily as a tool for verification and validation (V&V) [1]-[2]. This is because comparison of linear growth rates from a known equilibrium profile provides one of the few ways to quantitatively benchmark large-scale plasma turbulence codes. In this regard, the flexibility of the code is advantageous because it allows it to be used to verify a wide variety of codes, including codes that were not considered when it was designed, provided the code being verified displays a linear growth phase. However, in addition to being useful for benchmarking, an eigenvalue solver can be used for physics studies, for instance calculating the dispersion relation of a given set of model equations. In this regard flexibility is of paramount importance, since such models are likely to be specific to particular problems, hence change as different problems are considered.

Most of the concepts used in the development of ArbiTER are based on a preexisting eigenvalue code, 2DX [3]. Features borrowed from 2DX include the equation parser (input format for entering equations), the input format for entering data such as profile functions, the use of the SLEPc [4] sparse eigenvalue solver, and many of the routines used for managing sparse matrices. The ArbiTER code extends this by adding a topology parser (input format for entering topologies) as well as associating functions and variables with specific user-defined computational domains. Thus, whereas 2DX is limited to solving problems in a 2-D x-point topology or some subset thereof, the ArbiTER code can handle any topology that can be represented using the current version of the topology parser.

The plan of our report is as follows. In Sec. II we describe the new code. Sec. III describes the physics models used for verification. Sec. IV is the main body of our report. Here we present a series of verification "benchmark" comparisons with either analytic results or pre-existing codes. A summary is given in Sec. VI. Additional details are given in Appendices.

II The ArbiTER code

The ArbiTER code is an eigensolver for linear partial differential equations in nearly arbitrary geometry and topology. The flexbility of this code derives from its use of an equation and topology parser to read model equations from input files. The structure of these parsers is described in the following sections.

A Code structure

The ArbiTER code builds and stores an eigenvalue problem through successive manipulation of sparse matrices. This ultimately results in a generalized eigenvalue problem of the form:

$$Ax = \lambda Bx \tag{1}$$

Once the sparse matrices A and B have been constructed, the code passes these matrices to the SLEPc [4] eigenvalue solver. Once SLEPc returns a list of eigenvectors, each entry in the vector is assigned coordinate positions by the ArbiTER code according to its position within the relevant computational domain, prior to being saved to a file.

The construction of these sparse matrices is regulated by three input files. The equation file regulates the manner in which a small initial set of sparse matrices will be manipulated. The actual construction of the initial building blocks used by the equation parser is determined by the grid file, which provides profile functions and other data specific to an instance of the problem to be solved, and the topology file, which determines the connectivity of each computational domain and defines the basic operators used to calculate derivatives via finite difference methods.

The lack of hard coding (model equations or topology determined by source code) is possible because the source code is devoted mainly to instructions for executing elementary operations. Because these elementary operations are used in many different models, and in many different terms in a given model, this makes verification of the ArbiTER source code itself very simple; verification tests done for one model also build confidence in the code as a whole. Of special importance for both ArbiTER and its predecessor 2DX is the incorporation of efficient routines for adding and multiplying sparse matrices. These routines are both algorithmically fast (order n to add sorted matrices, order $n\ log(n)$ to sort matrices, order $n\ log^2(n)$ to multiply sorted matrices). Because of this, and because solving the eigenproblem is much more computationally intensive than generating it, the advantage in run time of a hard-coded equation set (i.e. explicitly defining each matrix element in the source code) over soft-coding (i.e. progressing through a series of matrix operations) is negligible.

The topology parser is possible, in part, through adaptation of the data structures used to store sparse matrices. In essence, the topological connectivity of a computational domain, as well as any differential operators defined on that domain, can be stored as sparse matrices. Only a small amount of additional

information (such as assigning grid coordinates to each point) is needed to define a computational domain in its entirety.

B Equation parser

The equation parser in ArbiTER is nearly identical to that used in 2DX. This fact is used in Sec. IV.A to verify the ArbiTER code by emulating 2DX; since the equation parsers are so similar, it is relatively straightforward to convert structure files between the two formats.

The equation file consists of three major parts. The first is the label list, which determines how the grid file will be parsed. Thus, each label (an identifier found at the beginning of each section of the grid file) is followed by a variable type (integer, real, real function, complex function), an index number, and a domain (in the case of functions).

The second is the formula language, which governs construction of systems of equations from simpler building blocks. For instance, if we take the second equation in the resistive ballooning model,

$$\gamma \delta N = -\delta v_E \cdot \nabla n_0 \tag{2}$$

this might be coded as

```
gg*(1+0j)*N=(-1+0j)*kbrbpx*n0p*PHI
```
(3)

where `gg` is the structure file notation for the eigenvalue γ, complex constants are in parenthesis, `N` $= \delta N$, and `PHI` $= \delta\phi$ are field variables, and `kbrbpx` and `n0p` represent a function and input profile, respectively.

The third is the element language. The element language allows complicated functions or operators to be derived from simpler building blocks. This is accomplished through a series of at most binary operations. Thus, an operator of the form:

$$\nabla_\parallel = \jmath \partial_y^u \tag{4}$$

might be represented through a series of instructions such as the following:

`xx=xjac`	load function $\jmath$ into function buffer
`xx=interp*xx`	multiply function buffer by operator interp
`tfn1=xx`	place result in function tfn1
`yy=ddyu`	load operator ∂_y^u into operator buffer
`yy=tfn1*yy`	multiply opertor buffer by function tfn1
`op1=yy`	place result in operator op1 ($\nabla_\parallel$)

(5)

One significant improvement over the 2DX equation parser is the ability to apply arithmetic operations to scalar quantities. This is facilitated by the fact

that functions in ArbiTER are assumed to have a variable number of entries depending on the size of the computational domain they occupy; thus, a scalar is simply a function over a "domain" consisting of a single element. Another improvement is the ability to apply user-defined operators directly to functions; in 2DX operators could only be applied to variables in the formula language, so the derivative of a function, for instance, would need to be provided as a separate profile function. The utility of applying operators to functions is demonstrated in Eq. 5, as the input function `xjac` (j) is interpolated using the operator `interp` to calculate its values on a staggered grid.

C Topology parser

The singular innovative feature that distinguishes the ArbiTER code is its topology parser. The topology parser is responsible for creating computational domains for each variable or profile function referenced by the equation parser. It is also responsible for generating the elementary differential operators used to calculate derivatives. The latter represents a major break from the 2DX code, in which elementary differential operators are built-in.

The topology parser allows the user to define five types of elements: bricks, linkages, domains, renumbers, and operators. Of these, bricks, renumbers and operators currently have only a single variant, but the syntax of the topology parser permits additional variants to be added in future versions of the code. Linkages and domains are currently the most diverse objects definable using the topology parser. Definitions are executed sequentially, so any previous definitions can be referenced by subsequent objects.

Bricks (sometimes also known as blocks) are simple Cartesian grids. Brick definitions have a variable number of arguments, each of which references an integer variable specifying one of its dimensions. Since the number of arguments is variable, the number of dimensions is arbitrary; the data structures used in the ArbiTER code do not result in any intrinsic limit on dimensionality.

Linkages are sparse matrices specifying a set of adjacent points between two blocks or domains.

Domains are sets of points on which a function or variable can be defined. Domains are normally created by combining one or more blocks and a number of linkages. In this case, the linkages specify how the blocks connect to each other, and are appended to the connectivity matrices of the domain after its constituent blocks have been merged into a single object. In addition, domains can also be generated by convolving two existing domains, i.e. performing an outer product of the grid points of those domains. Convolved domains are useful in kinetic problems because they allow a higher dimensional space to inherit the topological properties of a lower dimensional space.

Renumbers alter the sequence of grid points inside a domain. Because functions are loaded from the grid file sequentially, the order of grid points is important in order to ensure that profile functions are interpreted as intended. Since the normal method for constructing domains can result in grid points being numbered in an unintuitive manner, renumbers fix this problem. In ad-

dition, the syntax for renumbering allows additional numbering schemes to be introduced in later versions of the code.

Operators are the sparse matrices used to calculate derivatives by the equation parser. An operator is created by combining one or more linkages. In addition, the identity matrix of a domain is available as an implied linkage connecting the domain to itself, even if no such linkage has been declared.

In addition, the topology file also contains a section for processing integer inputs. The syntax used in this section is based on the element language of the equation parser, but using integer rather than complex operands. The purpose of this capability is to calculate the sizes of sub-domains (blocks) and offsets of linkages based on a limited number of integer inputs. Thus, in an x-point topology, the user can specify the grid position of the x-point, then allow the integer language to calculate the sizes of the scrape-off layer, edge, and private SOL regions. This reduces the number of integer inputs, as well as allowing the topology parser to enforce internal consistency between different components.

III Verification models and topologies

A The resistive ballooning model

The resistive ballooning model is used in the 2DX emulation test, and is based on a prior benchmark of the 2DX code [5]. The model equations are as follows:

$$\gamma \nabla_\perp^2 \delta\phi = \frac{2B}{n} C_r \delta p - \frac{B^2}{n} \partial_\parallel \nabla_\perp^2 \delta A \tag{6}$$

$$\gamma \delta n = -\delta v_E \cdot \nabla n \tag{7}$$

$$-\gamma \nabla_\perp^2 \delta A = \nu_e \nabla_\perp^2 \delta A - \mu n \nabla_\parallel \delta\phi \tag{8}$$

where

$$C_r \equiv \vec{b} \times \kappa \cdot \nabla \tag{9}$$

$$\delta v_E \cdot \nabla Q \equiv -i \frac{k_b (\partial_r Q)}{B} \delta\phi \tag{10}$$

The differential operators in this model are defined in field-line following coordinates. These are described in greater detail in Appendix A.

For the 2DX emulation test, the topology parser file describes an x-point geometry. This can be reduced to any subset by choice of parameters defining the size of each block. The topology used for 2DX emulation uses seven total blocks. One is used to provide an input format for periodic boundary conditions. Four are used to generate the computational domain. Two more are used to create an indented version of the computational domain, i.e. one missing one column of grid points from the right hand sheath. The indented domain is used to permit variables to be placed on a staggered grid, thus avoiding numerical issues associated with central difference methods for calculating derivatives. The layout of the main computational domain is shown in Fig. 1.

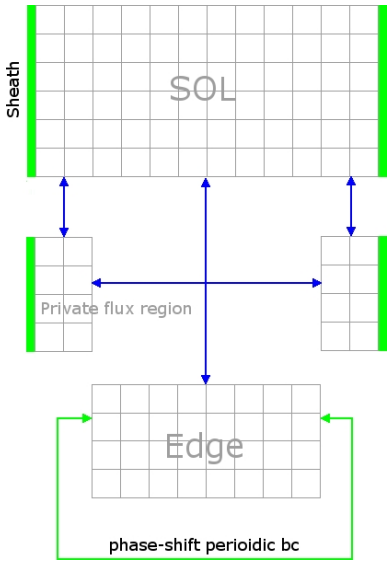

Figure 1: Computational domain for 2DX emulation topology. Blue arrows represent simple linkages, green arrows represent phase-shift periodic linkages.

B The parallel kinetic model

The parallel kinetic model consists of a modified 1-D Vlasov equation. Its purpose is to provide a simple test of ArbiTER's kinetic capabilities that can be benchmarked against analytic theory.

The model equations are as follows:

$$-\gamma \delta f = v \partial_x \delta f + \partial_v f_0 \partial_x \phi + \alpha \partial_v^2 \delta f \tag{11}$$

$$\gamma \phi = \mu_{eff} \partial_x^2 \phi - \mu_{eff} \int_v \delta f \tag{12}$$

The topology for this model consists of three domains: a real space domain, a velocity space domain, and a phase space domain. The velocity space domain is used only to input the equilibrium distribution function. The real space domain is used to calculate potential, and has periodic boundary conditions. Each of these is 1-D. The phase space domain is constructed by convolving real and velocity space. As such, it inherits the periodic boundary condition from real space.

The model equations shown calculate electric fields by relaxation; in order to convert Eq.12 into eigenvalue form, both parts of the equation have been moved to the right hand side, then multiplied by a large number μ_{eff}. Thus, so long as μ_{eff} is chosen to be much larger in magnitude than the eigenvalue, the desired equation will be approximately zero. This is the desired constraint equation.

A velocity diffusion term $\alpha\partial_v^2\delta f$ is added to Eq.11 in order to avoid numerical instabilities. Without this term, each point in velocity space creates a pole in the dispersion relation. By adding sufficient amounts of velocity space diffusion, the poles are smoothed out and a realistic dispersion curve results. Care must be taken not to set the parameter α too high, as it will also damp physical modes of the system.

C The thermal diffusion model

The thermal diffusion model is a very simple fluid model for benchmarking the capability to solve problems in double-null geometry. The model equation is:

$$\gamma T = \kappa_\parallel \nabla_\parallel^2 T + \kappa_\perp \nabla_\perp^2 T \tag{13}$$

where $\nabla_\perp$ is defined identically to that used in the 2DX emulation model and $\nabla_\parallel^2 = \partial_\parallel \nabla_\parallel$ in the same model. These operators are described in detail in Appendix A.

Because this model is intended for a benchmark in double x-point topology, the topology parser file is different than that used in the resistive ballooning model. This is because the double x-point topology contains six distinct topological regions (edge, upper and lower private SOL, left and right SOL, and intermediate SOL) compared to only three for the single x-point case (edge, SOL, private SOL). In addition, the topology is organized differently; rather than use bricks that each contain only a single topological region (in the single x-point case the private SOL is divided in two, but no brick contains parts of more than one region), instead the domain is divided into six bricks, each containing points from two to three separate topological regions. This layout is possible due to special topology parser capabilities, such as the ability to create double offset linkages (i.e. only some of the points on the matching faces are linked) and dummy linkages (i.e. none of the points on the matching faces are linked). The layout of this topology is shown in Fig. 2.

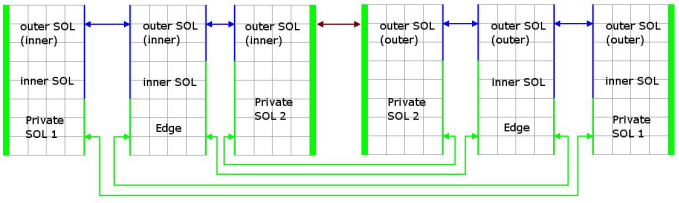

Figure 2: Topology in the double null benchmark case. Blue arrows represent simple linkages, green arrows represent phase-shift periodic linkages, brown arrows represent dummy linkages.

IV Verification tests

A Test 1: 2DX emulation

The most basic test of ArbiTER functionality is to verify its ability to emulate the 2DX eigenvalue code. Since both use nearly the same equation parser, and since the topology used in 2DX is straightforward to define using the topology parser, the ArbiTER code should be able to reproduce results from 2DX. For this purpose, the resistive ballooning model, which was used to benchmark 2DX, is used.

The actual simulation test was done using Eqs. 6-9. The geometry used is a shearless annulus with periodic boundary conditions. This is constructed as a subset of the 2DX emulation topology by setting the last closed flux surface outside the outermost flux surface, so that only closed flux surfaces are calculated. The parameters determining the boundaries of the edge region are set to the boundaries of the domain, so that no private flux region is calculated.

The results from this were compared to prior results from a benchmark of the 2DX code against BOUT [6] and analytic theory. The results of this are shown in Fig. 3. The results from the other codes involved in the benchmark are not shown, as the purpose of this test is merely to determine whether ArbiTER can emulate 2DX (for better or for worse) and since the previous benchmark showed accurate solutions from 2DX.

The results are shown in Fig. 3.

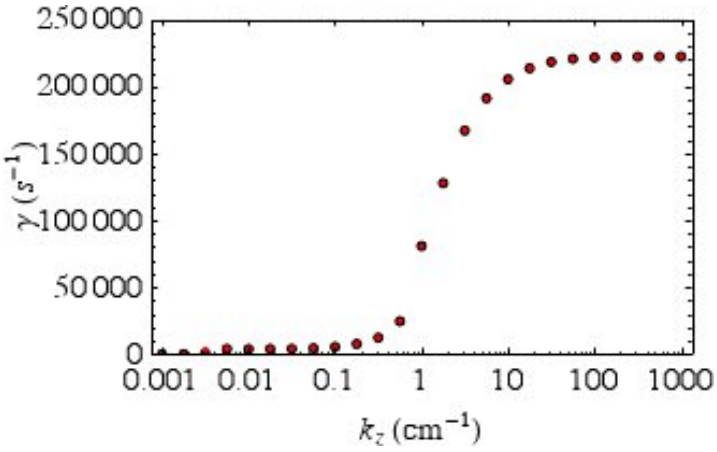

Figure 3: Comparison of ArbiTER and 2DX results for a resistive ballooning model. Black dots are 2DX results, red dots are ArbiTER results.

B Test 2: Langmuir waves

This test uses the model in Sec. III.B to calculate the dispersion relation for Langmuir waves. This model exercises ArbiTER's capabiity to create convolved

domains, as well as operators linking these domains in a regular manner. Thus, despite its simplicity, it exercises all of the features needed to calculate eigenvalues in a full kinetic model.

The domains used in this model contain 52 points in velocity space and 3 points in real space, for a total of 156 points in phase space. The size of the real space was chosen because three points under periodic boundary conditions permit only a single nonzero wavenumber. This allows the wavenumber to be controlled based only on grid spacing, without the need to isolate a particular mode of interest from a spectrum of subdominant modes. In addition, numerical dispersion can be corrected for by calculating the permitted effective wavenumber for a particular grid spacing based on properties of the finite difference operators involved, rather than from an approximate formula valid only for well-resolved modes.

The derivative of a Maxwellian was provided as an input profile function. A moderate value for velocity space diffusion was also used. A multiplier on the input distribution function was used to vary density. Since plasma frequency varies with density, this results in a dispersion curve that can be compared with theory. A similar result can also be obtained by varying grid spacing.

The results of this were compared with analytic theory. One model used was an approximation based on fluid theory. The other used the full plasma response function (Z) to calculate an exact dispersion function. A comparison between ArbiTER results and each of these models is shown in Fig. 4

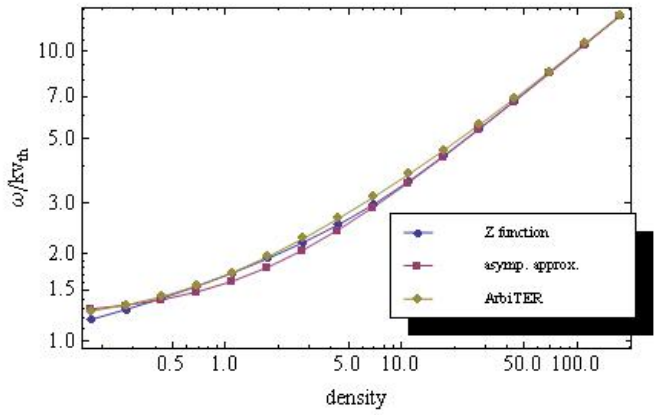

Figure 4: Comparison of ArbiTER results to analytic theory for a Langmuir wave problem.

C Test 3: double null geometry

The ArbiTER code was used to solve a heat conduction problem in double-null geometry. This geometry was based on data from Alcator C-Mod [8]. Unlike

the sort of heat conduction problem one would normally encounter in a tokamak edge plasma, in this case the problem was simply that of thermal relaxation with fixed-value boundary conditions on all sides. This has the advantage that it is both simple and it constitutes an eigenvalue problem, with the rate of relaxation from the starting to the equilibrium profile being the eigenmode.

These results were then benchmarked by comparing them to results from the UEDGE code [7]. In these cases, UEDGE was given the same problem with a perturbed starting profile and used to calculate the rate at which that profile relaxes. The deviations from uniform temperature after a sufficiently large number of decay times can then be compared to the least-damped eigenmode using ArbiTER. This provides a comparison of both decay rates and mode structure.

In the course of this study, a significant complication was uncovered. Because UEDGE is an implicit time-stepping code, the apparent decay rate in the UEDGE result depends on the size of the time step chosen. In particular, if $\gamma\delta t$ is close to 1, where γ is the decay rate, then the code will greatly underestimate the decay rate. In order to compensate for this without the need to use extremely small time steps, UEDGE was run multiple times for each test case with a different time step size in each case. These results were then used to calculate an asymptotic value using Richardson extrapolation.

Another complication arises because the model equations in UEDGE differ from the model used in ArbiTER by a factor of 3/2 in the left-hand side of Eq. 13. As a result, all UEDGE results have to be normalized by multiplying by 3/2 before comparison to ArbiTER.

Growth rates were compared for $\kappa_\perp = 1$ for three widely divergent values for $\kappa_\parallel$. The results of this comparison are shown in Fig. 5. In addition, the eigenfunctions for the case $\kappa_\parallel = 10^6$ were compared; these were of particular interest because the eigenfunction spans both x-points. The results of this comparison are shown in Fig. 6.

The results for the eigenvalues show close agreement between the two codes once appropriate normalizations and other corrections are taken into account. However, the eigenfunctions show only qualitative agreement. The source of quantitative disagreement is not known at this time. It is worth noting that the UEDGE eigenfunction shows some resemblance to the square of the ArbiTER eigenfunction, but no explanation of this resemblance is forthcoming. In spite of this disagreement, the agreement in eigenvalues suggests that the two codes are producing consistent answers for the same problem.

V Extensions and additional capabilities

A Parallel solver

Work has begun on incorporating the parallel computing capabilities of the SLEPc package into ArbiTER. To test the potential of this line of research, a stand-alone parallel solver based on the SLEPc package has been built. The ArbiTER code is then used to create matrices for SLEPc to solve. While this is

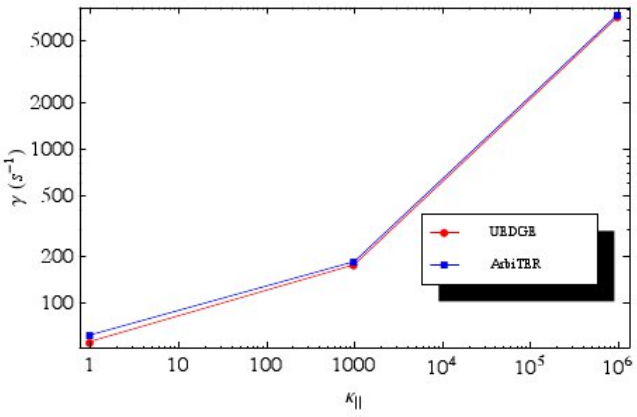

Figure 5: Growth rate trends in the UEDGE and ArbiTER double null test cases. Both cases were run with normalized $\kappa_\perp = 1$.

much less convenient than the fully integrated eigenmode solving capabilities of the serial version of ArbiTER, it is still sufficient to estimate code performance using parallel routines.

Two different problems were studied using the stand-alone solver. One is based on an ELM benchmark [9] that was previously studied using 2DX [10]. This problem was chosen because its high resolution requirement resulted in large matrices for which a parallel solver would be most desirable. In addition to testing the parallel solver, this also served as a benchmark of ArbiTER against 2DX, since the solution using the 2DX code was already known. Moreover, many properties of this set of equations (most notably the use of a nontrivial left-hand side matrix) are shared by many other physics models used in the plasma edge and scrape-off layer.

The other problem was a simple heat diffusion problem in a three-dimensional box. This was chosen for its simplicity, its ability to produce large problems with a small grid file (since there were no profile functions, only the size of the box needed to be entered), and its lack of a left-hand side matrix. As we will see, these properties resulted in dramatically different scaling properties with regards to number of processors.

The results from the ELM case are shown in Fig. 7. This shows a significant reduction in wall-clocktime when a single processor is added, but less significant reductions when additional processors are added. This indicates some barrier to efficient parallelization.

The heat diffusion problem, on the other hand, displayed excellent scaling with increased numbers of processors. The results from this study are shown in Fig. 8. In this case, scaling with number of processors is nearly linear, indicating extremely efficient parallelization of this problem.

The difference between these can be explained by the presence or absence of a

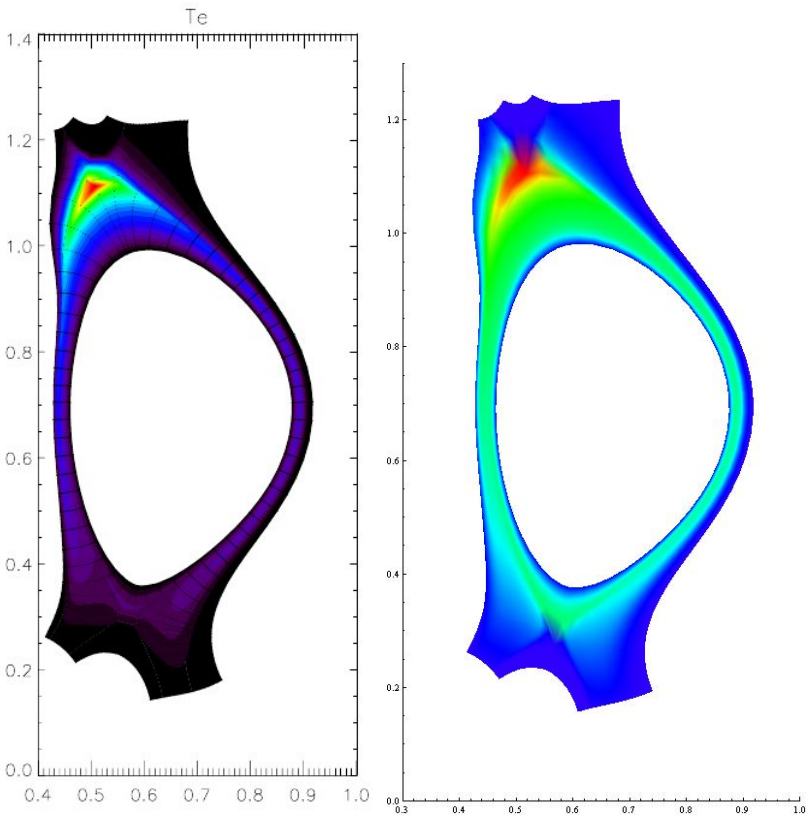

Figure 6: Sample eigenmodes from both ArbiTER and UEDGE for the double null test case. In this case, $\kappa_\perp$ is 1 and $\kappa_\parallel$ is 10^6. Note that these plots were generated with different color coding.

nontrivial matrix on the left-hand side of the equations. Having a left-hand side matrix means that, at each iteration of the underlying Krylov-Schur method, the left-hand side matrix must be solved. The default method for solving this matrix that is used by SLEPc does not parallelize, creating a significant time overhead for large problems. Options can be used to select other methods that do parallelize well, however, these methods are slower overall for problems of this size. This suggests that for problems with a left-hand side matrix, there exists a minimum number of processors before parallel methods offer a significant advantage over serial methods. Alternately, there exists a minimum problem size before parallel methods demonstrate significant value. The exact boundaries of these different regimes, or the exact options needed to optimize in these regimes, were not determined in the course of this project. However, knowledge of their existence represents useful experience for future reference.

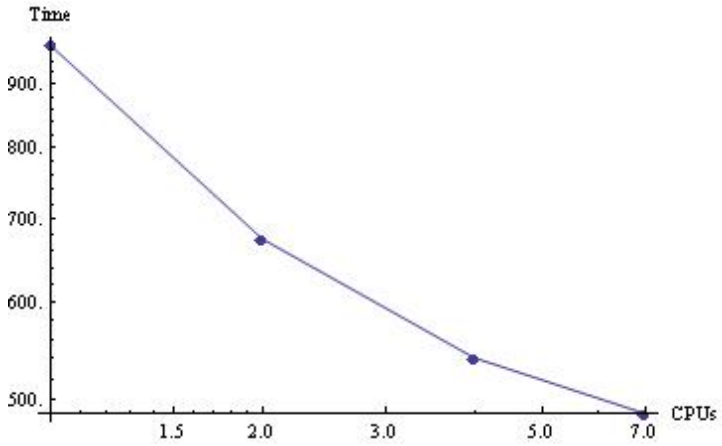

Figure 7: Scaling of time to solve an ELM problem in seconds vs. number of processors.

VI Summary

Successful benchmark tests have been carried out between the new eigenvalue code ArbiTER and a number of existing codes, such as 2DX and UEDGE, as well as verification tests against analytic theory. These tests demonstrate the effectiveness and accuracy of the new code, as well as its versatility. These results are a step towards demonstrating the potential to create a near-universal PDE eigensolver.

The ArbiTER code is notable in its use of parsers to define both equation sets and topology. The equation parser, while largely inherited from 2DX, is an important capability in itself; it permits rapid customization of model equations, making the code largely physics-independent and thus capable of rapid adaptation as physics models improve, or of application to completely different types of problems. The topology parser, in turn, permits customization of the topology of each computational domain. In addition to simply altering the connectivity of regions within a domain, the topology language also permits domains to be constructed by convolution of other domains. This is essential for solving kinetic problems, as such problems involve a phase space that must inherit the topological properties of its accompanying real space, as well as requiring a simple and consistent method for mapping functions and variables between the two spaces (i.e. integrating velocity or convolving with a distribution function). Such capabililty is provided by the current version of the code.

In addition to its current capabilities, the topology parser is highly upgradeable. Because each instruction type can support an arbitrary number of subtypes, and because each instruction accepts a variable number of arguments, new features can be added to the topology language without sacrificing reverse compatibility with regard to topology files. This capability may be used in the

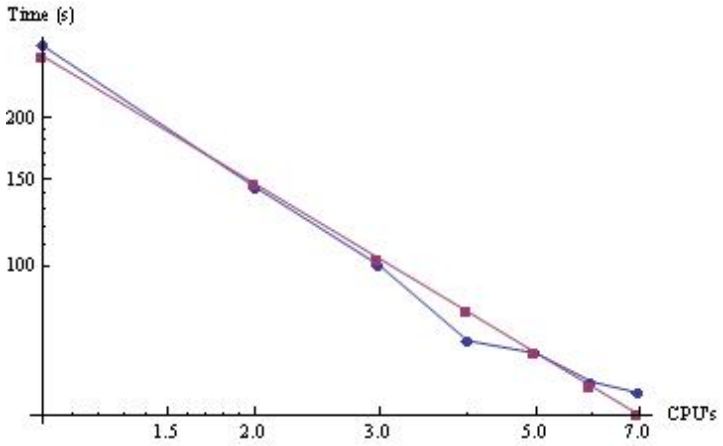

Figure 8: Scaling of time to solve a 40x40x40 heat diffusion problem vs. number of processors. Blue curve is actual timing data, red curve is a best fit. Scaling exponent for the red curve is .859676.

future, for instance, to introduce different node numbering schemes, or to add finite element analysis capabilities to the code.

ArbiTER currently uses the SLEPc sparse eigensolver package. This package, as its acronym suggests, is capable of large-scale parallel operation. Unfortunately, the code routines for interfacing with the parallel capabilities of this package are different than those used in serial operation, so significant code restructuring is required to fully integrate this capability. However, the use of SLEPc as a stand-alone eigensolver using ArbiTER to generate matrices has demonstrated the feasibility of such operation, and fully parallel operation requires only the development of code to pass data directly to the parallel SLEPc routines rather than through a data file. Since matrix setup time is small compared to that required to solve the eigensystem, significant speed improvements can be made without the need to parallelize the ArbiTER kernel. The actual construction of such an integrated code is left for future work.

Appendix A: Coordinate systems and differential operators

The differential operators used in the 2DX emulation test are defined using field-line-following (FLF) coordinates. These are defined as follows [11]:

$$\begin{aligned} x &= \psi - \psi_0 \\ y &= \theta \\ z &= \zeta - \int_{\theta_0}^{\theta} d\theta \nu \end{aligned} \tag{14}$$

where θ_0 is an arbitrary constant. Invoking toroidal mode expansion, it can be shown that this is equivalent to assuming a mode representation of the form

$$\Phi = \Phi_{FLF}(x,y) exp\left(in\zeta - in\int_{\theta_0}^{\theta} d\theta \nu \right) \tag{15}$$

where $\Phi_{FLF}(x,y)$ is the function that is being solved for numerically. When $k_\parallel \ll k_\perp$, this ensures that $\Phi_{FLF}(x,y)$ is slowly varying even for large n. Basically, the y dependence corresponds to $k_\parallel$ and the fast θ dependence has been extracted into the phase factor in Eq. 15. In these coordinates, the representation for the differential operators is

$$\nabla_\parallel = \frac{1}{\nu R}\frac{\partial}{\partial y} \tag{16}$$

$$\nabla_\perp^2 = -\frac{1}{J}(k_\psi - i\partial_x^a) J (k_\psi - i\partial_x^b) - k_b^2 \tag{17}$$

where

$$k_b = -nB/RB_\theta \tag{18}$$

$$k_\psi = -n|\nabla\psi| \left(\frac{\nu \nabla\theta\cdot\nabla\psi}{|\nabla\psi|^2} + \int_{\theta_0}^{\theta} d\theta \frac{\partial \nu}{\partial \psi} \right) \tag{19}$$

$$\partial_x^a Q = \frac{\partial}{\partial\psi}|\nabla\psi| Q \tag{20}$$

$$\partial_x^b Q = |\nabla\psi| \frac{\partial}{\partial\psi} Q \tag{21}$$

$$J^{-1} = \nabla\psi \times \nabla\theta \cdot \nabla\zeta \tag{22}$$

The solution Φ in the FLF representation is periodic in θ when Φ_{FLF} obeys the phase-shift-periodic boundary condition

$$\Phi_{FLF}(y=0) = \Phi_{FLF}(y=2\pi) e^{-2\pi i n q} \tag{23}$$

where

$$q = \frac{1}{2\pi}\int_0^{2\pi} d\theta \nu \tag{24}$$

References

[1] P.J. Roache, Verification and Validation in Computational Science and Engineering, Hermosa Publishers, Albuquerque, NM (1998).

[2] W.L. Oberkampf and T. G. Trucano, Progress in Aerospace Sciences **38**, 209 (2002).

[3] D. A. Baver, J. R. Myra, M. V. Umansky, Comp. Phys. Comm. doi:10.1016/j.cpc.2011.04.007 (2011).

[4] http://www.grycap.upv.es/slepc

[5] www.lodestar.com/research/vnv/AppB%20eccvvrb3h.pdf

[6] M.V. Umansky, X.Q. Xu, B. Dudson, L.L. LoDestro, J.R. Myra, Computer Phys. Comm. **180**, 887 (2009).

[7] T.D. Rognlien, J L. Milovich, M. E. Rensink, and G.D. Porter, J. Nucl. Mater. 196-198, 347 (1992).

[8] I. H. Hutchinson, R. Boivin, F. Bombarda, P. Bonoli, S. Fairfax, C. Fiore, J. Goetz, S. Golovato, R. Granetz, M. Greenwald, S. Horne, A. Hubbard, J. Irby, and B. L. B. LaBombard, E. Marmar, G. McCracken, M. Porkolab, J. Rice, J. Snipes, Y. Takase, J. Terry, S. Wolfe, C. Christensen, D. Garnier, M. Graf, T. Hsu, T. Luke, M. May, A. Niemczewski, G. Tinios, J. Schachter, and J. Urbahn, Phys. Plasmas **1**, 1511 (1994).

[9] H. Zohm, Plasma Phys. Controlled Fusion **38**, 105 (1996).

[10] www.lodestar.com/research/vnv/AppI%20eccvvelm5h.pdf

[11] Linear Analysis LRC report,
http://www.lodestar.com/LRCreports/Linear%20Analysis%20LRC%20Report.pdf