Parallelization of DEBS

By

Brian Cornille

A Senior Thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor's of Science

(Engineering Physics)

at the

UNIVERSITY OF WISCONSIN – MADISON

2014

Date of final oral examination: December 16th, 2014

Abstract

This thesis presents the parallel development and testing of the DEBS resistive magnetohydrodynamics (MHD) code [D. D. Schnack et al., "Semiimplicit magnetohydrodynamic calculations", Journal of Computational Physics **70**, 330 (1987)]. Parallelization of DEBS will allow for faster turnaround of calculations, and greater numerical resolution in reversed-field pinch (RFP) simulations. Since the semi-implicit time advance of DEBS requires solving a matrix equation where the matrix is narrow-banded, the parallel implementation is designed around the use of the parallel linear algebra library ScaLAPACK. These matrix equations lie across the radial mesh of the cylindrical domain, and that mesh needs to be distributed across multiple processors. The parallel implementation for DEBS is a balance between minimizing the changes to the code and the maximizing effectiveness of the parallel computations. Computations have been completed to measure the strong and weak scaling properties of parallel DEBS. Based on the strong scaling study, the number of radial points per processor is best kept in the range $\frac{N_r}{P} \geq 50$, where N_r is the radial mesh size and P is the number of processors used. The weak scaling study indicates that for radial mesh sizes typically run for RFP simulations with DEBS, the limiting factor of parallel execution time is due to the latency associated with inter-processor communication. Thus, for best performance a single shared memory compute node should be used for production level calculations. A production-scale RFP application is used to exercise the new parallel implementation of DEBS. Three different mesh sizes are considered with identical initial conditions. The radial resolution is either similar to or greater than that of typical simulations run with

the serial version of DEBS. Temporal traces of reversal parameter and maximum Prandtl number show differences over a full sawtooth cycle, hence numerical convergence is not achieved.

Acknowledgements

First and foremost, I would like to thank my research advisor and mentor, Professor Carl Sovinec. This project would not have been possible without his guidance and expertise. Thank you for your consistent support and advisement throughout my undergraduate career.

I would also like to express my appreciation for my thesis committee, including Professor Carl Sovinec, Professor John Sarff, and Professor Greg Moses. Professor John Sarff was instrumental in determining the subject of this thesis, and was also gracious enough to provide funding support for one summer of this work. Professor Sarff and Professor Sovinec also served on the committee for my proposal of this research project. I thank you all for taking the time not only to review, but to take a sincere interest in the work that I have done.

Dr. Josh Reusch has been another important individual that has aided in my success on this project. He provided me with my original copy of DEBS and has provided useful instruction on running it. Dr. Josh Reusch and Dr. Craig Jacobson provided me with the input files for the test cases I ran with DEBS. A special thank you to Dr. Josh Reusch for helping me to compile data from several of my DEBS output files so that it could be included in my thesis.

This research was performed in part using the computer resources and assistance of the UW-Madison Center For High Throughput Computing (CHTC) in the Department of Computer Sciences. The CHTC is supported by UW-Madison, the Wisconsin Alumni

Research Foundation, the Wisconsin Institutes for Discovery, the National Science Foundation, and the Advanced Computing Infrastructure, and is an active member of the Open Science Grid, which is supported by the National Science Foundation and the U.S. Department of Energy's Office of Science. In particular I thank Neil Van Lysel for promptly answering my questions about the high performance computing cluster and for compiling the ScaLAPACK library on that system.

Contents

List of Figures

Chapter 1

Introduction

The main subject of this thesis is the parallelization of a code called DEBS. DEBS is a program that can be used to simulate macroscopic plasma dynamics in cylindrical geometry. An introduction to this code and some motivation for making these changes are discussed in section 1.1. Further discussion of the physical system modeled by DEBS is presented in section 1.2. To give context for the strategies employed for parallelization, background on the field of parallel computing is described in section 1.3. The numerical methods used within DEBS are introduced in section 1.4. The specifics of these numerical methods provide the basis for extending DEBS to work as a parallel computer code.

1.1 DEBS

The DEBS program solves the non-linear, resistive magnetohydrodynamics (MHD) equations for the time evolution of magnetic vector potential, fluid velocity, pressure, and density.[1] These equations govern the fluid-like behavior of plasma in the limits of low frequency and long wavelength. Plasma dynamics are important in the study of both astronomical phenomenon and magnetic confinement for fusion applications. The Madison Symmetric Torus (MST) group uses DEBS for calculations to compare with their

experiment. The MST experiment operates with a reversed-field pinch (RFP) magnetic configuration in a toroidal vacuum vessel having a reasonably large aspect ratio.[2, 3] DEBS assumes a problem geometry of a periodic cylinder, which models the plasma dynamics of an RFP quite well if the aspect ratio is not too small.

Periodic instability due to inductive current drive, known as sawtooth oscillations, is one important phenomenon in RFP plasma physics that DEBS is capable of replicating. As a member of the MST group, Dr. Josh Reusch examined an ensemble of over 5,000 sawtooth events that occurred in MST. He also ran DEBS and was able to produce an ensemble of 23 sawtooth events to compare with. This small number of simulated sawtooth events consumed nearly a year of real time to compute.[4]

DEBS was originally written in the mid 1980s, but has gone through various revisions since then. Over this time, several updates have been made to the physics and algorithms to improve the accuracy of the program and also, to a lesser degree, improve its efficiency. The core structure and flow of DEBS is still based on programming that was efficient for machines that were top-of-the-line in the 1980s, namely vector processors. These types of processors were able to execute many calculations per second because they worked similar to assembly lines. As long as one piece of data was not dependent on a calculation within the same programming loop, then multiple data sets could move through the processor simultaneously. Additionally, DEBS was designed so that many intermediate calculations are done in place rather than saving values in memory. This creates a very efficient program, but the way the program is currently written does not allow for the code to take advantage of current supercomputers, which rely on distributing the problem across many computer cores to leverage more computing power.

1.2 Magnetohydrodynamics

A frequently used theory for describing the behavior of plasma is MHD. This theory treats the plasma as a continuous fluid that is quasi-neutral and electrically conducting. Quasi-neutrality means that the forces from net charge density are negligible compared to other forces, such as the Lorentz force or forces from pressure gradients. The fact that plasma is electrically conducting means that its fluid behavior is additionally coupled to electrical and magnetic fields that are present. One form for the MHD equations is

$$\frac{\partial \mathbf{A}}{\partial t} = \mathbf{v} \times \mathbf{B} - \eta \mathbf{J} \tag{1.1}$$

$$\rho \frac{\partial \mathbf{v}}{\partial t} = -\rho \mathbf{v} \cdot \nabla \mathbf{v} + \mathbf{J} \times \mathbf{B} - \nabla P + \nu \nabla^2 \mathbf{v} \tag{1.2}$$

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot \rho \mathbf{v} \tag{1.3}$$

$$\frac{\partial P}{\partial t} = -\gamma P \nabla \cdot \mathbf{v} - \mathbf{v} \cdot \nabla P + (\gamma - 1)[\nabla \cdot \underline{\underline{\mathbf{K}}} \cdot \nabla \left(\frac{P}{\rho} \right) + Q] \tag{1.4}$$

In equation (1.1) $\mathbf{A}$ is the magnetic vector potential defined by $\mathbf{B} = \nabla \times \mathbf{A}$ and $\mathbf{J}$ is the current density defined by $\mathbf{J} = {}^{1}/_{\mu_0} \nabla \times \mathbf{B}$. Other variables used throughout equations (1.1) to (1.4) include the plasma fluid velocity $\mathbf{v}$, magnetic field $\mathbf{B}$, mass density ρ, and pressure P. Additionally, in equation (1.1) η is the resistivity of the plasma, in equation (1.2) ν is the plasma's fluid viscosity, and in equation (1.4) γ is the isentropic expansion factor, $\nabla \cdot \underline{\underline{\mathbf{K}}} \cdot \nabla \left(\frac{P}{\rho} \right)$ is the thermal conduction term, and Q is the sum of all heat source terms.

DEBS uses a non-dimensional representation of equations (1.1) to (1.4) for its solution. Thus, plasma conditions are input using non-dimensional characteristic parameters, such as Lundquist number and magnetic Prandtl number. Lundquist number is defined by $S = {}^{v_A \tau_R}/_{a}$, where v_A is the Alfvén velocity, τ_R is the resistive diffusion time,

and a is the minor radius of the torus. Prandtl number is defined as $Pr_m = \frac{\nu\mu_0}{\eta}$ with viscosity ν and resistivity η. For many of the simulations run with DEBS, only equations (1.1) and (1.2) are used, setting pressure to zero and density uniformly to one. This subset, called zero-β, resistive MHD, is often sufficient to describe the current-density driven dynamics of interest for comparison to RFP experiments. Additionally, it can be quite tricky to produce a self-consistent and realistic simulation that fully includes pressure and density effects.

1.3 Parallel Computing

At present there are several well developed techniques for high performance computing. High performance computing methods tend to take advantage of multiple computing resources simultaneously. One of the most basic methods is shared-memory parallel computation. In this case multiple CPUs are used for processing, but they must all share the same memory, or RAM. These methods can be conceptually simple, but can be somewhat challenging to implement optimally. They are also strictly limited by the computer hardware on which they are usable. While it is no longer uncommon to have multiple CPUs sharing the same chip, shared-memory parallel computing can only scale to the number of processors that share the same memory space.

A more flexible method of high performance computing is distributed-memory parallel computation. This method takes advantage of multiple computers, or compute nodes, that are connected by some sort of network for communicating data. This method has the advantage of being able to access many computer resources simultaneously since, in principle, any number of computers can be connected. This also allows for a much

more flexible set of computer hardware to be used since compute nodes can be added or removed at will. One drawback of this kind of method is that computer-to-computer communication is much slower than the movement of data within one shared-memory node. This can lead to complications for many numerical methods that are routinely used in scientific computing. A well established system for preforming distributed-memory parallel computation is the Message Passing Interface (MPI).[5] Indeed, there are several computational libraries based on MPI that are readily available and are currently being used in scientific computing.

1.3.1 Measuring Parallel Performance

There are two main metrics for measuring parallel performance of codes. The first measure of scalability is called strong scaling. In strong scaling the computer resources are increased while the problem size is left constant. With perfect scaling one would see the speedup follow exactly with the number of processors. Speedup, S, is defined as

$$S = \frac{t_{seq}}{t_P} \tag{1.5}$$

where t_P is the execution time for P processors and t_{seq} is the execution time of the sequential, $P = 1$, code. Since perfect parallel performance is not generally achievable, strong scaling performance is often displayed in terms of efficiency.

$$E = \frac{S}{P} \tag{1.6}$$

Efficiency tends to be a more representative indicator of parallel performance since all graphs of speedup will show increases with more processors, making it necessary to carefully read the y-axis scale in order to judge a code's effectiveness. A code with

nearly ideal strong scaling performance will have E remain close to unity as P increases.

The other metric of measuring parallel performance is called weak scaling. With weak scaling the problem size is scaled with the increase in computer resources. The ideal case for weak scaling is that execution time will stay constant as the problem and number of processors increase together. This situation is much closer to practically achievable than the ideal case for strong scaling. Near ideal weak scaling behavior can be occur in the limit that total floating point operation time is much greater than the time required for communication between processors. With numerical techniques that require communication in their parallel implementation, this ideal behavior is usually only attainable with a large problem size per processor. This is due to data communication protocols being many orders of magnitude slower than single floating point operations. Even in the case of non-ideal weak scaling behavior there can be great gains compared to increasing the problem size for a serial calculation. While good strong scaling properties allow for reduced turnaround time of a given computation, weak scaling can be applied to increase numerical resolution for a given application.

1.4 Numerical Methods

The DEBS code utilizes several well-established numerical methods in order to achieve a robust algorithm for solving the MHD equations for a periodic cylinder.[1] For a periodic cylinder, the poloidal (θ), and axial (z) directions are both periodic. Thus, they are ideal candidates for spectral-type methods, which have extremely good spatial convergence properties for smooth functions. A pseudospectral algorithm for operations in the θ and z directions is used in DEBS. For the radial (r) direction, a finite differences method

evaluates spatial derivatives. For evolving the MHD equations in time, DEBS uses a semi-implicit time advance. This technique allows for numerically stable computations at time-step values that are much greater than possible with an explicit method, without the larger computational costs of a fully implicit method.

1.4.1 Pseudospectral Representation

In problems where one or more of the spatial coordinates can be considered periodic, as is true with DEBS, it is common to employ Fourier spectral methods rather than finite difference methods. This is because these methods have advantageous properties over finite differences. Arguably, the most important property of spectral methods is their order of accuracy. Finite differences methods can be designed to have orders of accuracy of first order, second order, etc. The error of these methods scale with $\frac{1}{N^p}$ for a pth order scheme with number of mesh points N. In comparison, spectral methods have errors that converge to zero with greater speed than any power of $\frac{1}{N}$ when they are applied to a *smooth function*[1]. They are often said to have *geometric convergence*.[6]

In DEBS, the periodic coordinates θ and z are treated spectrally. Rather than having a discrete mesh of points in these coordinates, variables in DEBS are represented as a set of complex finite Fourier coefficients $f_{m,n}$, which are defined as[2]

$$f_{m,n}(r) = \frac{1}{MN} \sum_{j=1}^{M} \sum_{k=1}^{N} f(r, \theta_j, z_k) e^{i(m\theta_j + n\frac{z_k}{R})} \tag{1.7}$$

[1]In this context we define a *smooth function* as a function that has continuous derivatives of any order. In mathematics, these are denoted as C^∞ functions.

[2]This is a non-standard convention for the discrete Fourier transform, but it is consistent with the programming of DEBS. This convention carries through to all equations related to the Fourier transform and Fourier components.

These coefficients can be transformed to a function in configuration space using equation (1.8), which has a discrete mesh with $\theta_j = \frac{2\pi(j-1)}{M}$ for $j = 1, 2, ..., M$ and $z_k = \frac{2\pi(k-1)R}{N}$ for $k = 1, 2, ..., N$, where R is the aspect ratio of the periodic cylinder.

$$f(r, \theta_j, z_k) = \sum_{m=-M/2+1}^{M/2} \sum_{n=-N/2+1}^{N/2} f_{m,n}(r) e^{-i(m\theta_j + n\frac{z_k}{R})} \tag{1.8}$$

In order for $f(r, \theta, z)$ to be real the Fourier coefficients must satisfy $f_{m,n} = f^*_{-m,-n}$. While variables are in their complex finite Fourier coefficient form, evaluation of derivatives in the spectral variables is an algebraic operation.

$$\frac{\partial f(r, \theta, z)}{\partial \theta} = \sum_{m=-M/2+1}^{M/2} \sum_{n=-N/2+1}^{N/2} -im f_{m,n}(r) e^{-i(m\theta + n\frac{z}{R})} \tag{1.9}$$

$$\frac{\partial f(r, \theta, z)}{\partial z} = \sum_{m=-M/2+1}^{M/2} \sum_{n=-N/2+1}^{N/2} -i\frac{n}{R} f_{m,n}(r) e^{-i(m\theta + n\frac{z}{R})} \tag{1.10}$$

Considering a single Fourier component of these derivatives, equations (1.9) and (1.10) can be expressed more simply.

$$\left(\frac{\partial f(r, \theta, z)}{\partial \theta}\right)_{m,n} = -im f_{m,n}(r) \tag{1.11}$$

$$\left(\frac{\partial f(r, \theta, z)}{\partial z}\right)_{m,n} = -i\frac{n}{R} f_{m,n}(r) \tag{1.12}$$

However, the operation of configuration space multiplication is nontrivial in a Fourier representation and requires a convolution sum over Fourier coefficients. These convolutions sums cost $O(N^2)$ operations. To avoid these convolutions the Fourier coefficients are converted to their configuration space representation using a Fast Fourier Transform (FFT), which only requires $O(N \log_2 N)$ operations.[7] This allows multiplications to be evaluated on the mesh in θ and z. The conversion from Fourier space to configuration space, which has limited resolution, to evaluate multiplication is what designates the

method used by DEBS as pseudospectral.[6] The main drawback of spectral methods appears when nonlinearities are present. Nonlinear terms can lead to aliasing errors, which generate modes with wavelengths shorter than the resolution of the spectral mesh. These errors can produce nonlinear numerical instabilities if they are not properly treated. The nonlinear terms that appear in the set of zero-β, constant density equations solved by DEBS are quadratic nonlinearities. Aliasing errors of quadratically nonlinear terms can be treated very simply by only including two-thirds of the available Fourier components in a particular spectral direction and setting the remaining modes to zero.[6] This anti-aliasing scheme is employed in DEBS with the FFT operations described above.

1.4.2 Finite Differences

Finite differences is a very common method for approximating derivatives in numerical solutions to differential equations. For example, given some spatial direction x and some variable ϕ that varies in x, the spatial derivative can be approximated

$$\left.\frac{\partial \phi}{\partial x}\right|_i \approx \frac{\phi_{i+1} - \phi_i}{\Delta x} \tag{1.13}$$

This example demonstrates what is called forward-differences with an evenly spaced mesh with spacing Δx. Many more sophisticated schemes for finite differences exist.

DEBS has many variables that vary spatially ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{J}$, $\mathbf{v}$, ρ, P), which have derivatives in r in various terms in the MHD equations. These variables exist on one of two radial meshes that are staggered from one another. The first mesh uses notation r_i and uses indices $i = 1, ..., N_r$. The variables A_θ, A_z, B_r, v_r, J_θ, J_z, ρ, and P are defined on this mesh. The second mesh uses notation $r_{i+1/2}$ with indices $i = 1, ..., N_r - 1$ and is defined by $r_{i+1/2} = (r_i + r_{i+1})/2$. The variables A_r, B_θ, B_z, v_θ, v_z, and J_r are defined on

this mesh. A pictorial representation of these meshes can be seen in figure 1. The mesh spacing is also defined on these two meshes by the relationships $\Delta r_i = r_{i+1/2} - r_{i-1/2}$ and $\Delta r_{i+1/2} = r_{i+1} - r_i$. This choice of distributing the variables provides a convenient

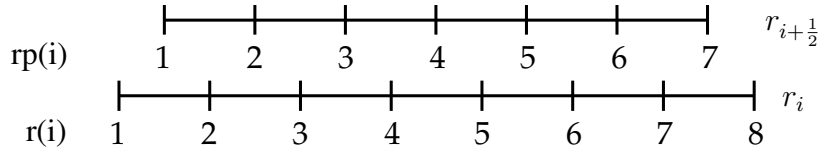

Figure 1: DEBS radial staggered mesh with $N_r = 8$. For pseudocode purposes these meshes may be referred to by r(i) and rp(i).

framework for evaluation of the radial derivatives that appear in various terms in the MHD equations. Namely, the evaluation of $\mathbf{B} = \nabla \times \mathbf{A}$ and $\mathbf{J} = \nabla \times \mathbf{B}$ on this mesh produces only nearest neighbor coupling when $\mathbf{J} = \nabla \times \nabla \times \mathbf{A}$ is evaluated numerically. This becomes important for the semi-implicit time advancement that DEBS uses. [1]

To demonstrate how this choice of meshes and their respective variables is apt, the finite-difference equations for specific terms of curl operator can be closely examined. Consider the equation for the θ term of $\mathbf{B} = \nabla \times \mathbf{A}$

$$B_\theta = \frac{\partial A_r}{\partial z} - \frac{\partial A_z}{\partial r} \tag{1.14}$$

Considering only the radial derivative, the finite differences on this mesh is

$$\left.\frac{\partial A_z}{\partial r}\right|_{i+\frac{1}{2}} \approx \frac{A_{z,i+1} - A_{z,i}}{\Delta r_{i+\frac{1}{2}}} \tag{1.15}$$

This derivative is evaluated for the $i + \frac{1}{2}$ mesh, on which B_θ is defined. Combining equations (1.11) and (1.15) allows for the full evaluations of B_θ. Within a code, the evaluation of $B_{\theta,i+\frac{1}{2},m,n}$ would be written as

```
bt(i,m,n) = -eye*n/R*ar(i,m,n)                    &
          - (az(i+1,m,n) - az(i,m,n))/drp(i,m,n)
```

where eye = **cmplx**(0.0,1.0). On an evenly spaced mesh, equation (1.15) behaves like central differences so it has second order accuracy. An equation analogous to equation (1.14) can be written for the θ term of $\mathbf{J} = \nabla \times \mathbf{B}$. The finite-difference approximation for the radial derivative in this equation is then

$$\left.\frac{\partial B_z}{\partial r}\right|_i \approx \frac{B_{z,i+\frac{1}{2}} - B_{z,i-\frac{1}{2}}}{\Delta r_i} \tag{1.16}$$

Combining equations (1.12) and (1.16) coded evaluation of $J_{\theta,i,m,n}$ would be

```
jt(i,m,n) = -eye*n/R*br(i,m,n)                    &
          - (bz(i,m,n) - bz(i-1,m,n))/dr(i,m,n)
```

Similar equations can be derived for any other differential operation as well. The full finite curl operators used in DEBS are given in Schnack et al. [1]. Based on these

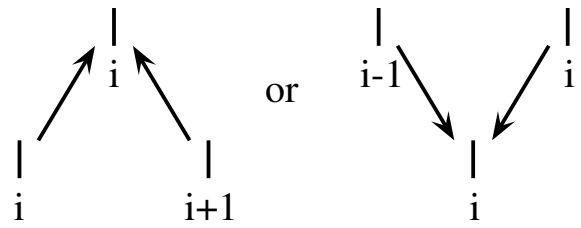

Figure 2: First derivative information dependencies.

equations one can see that information flow from single-derivative calculations flows from one mesh to the other.[3] A pictorial representation of this is shown in figure 2. This information dependency was important in creating the domain decomposition scheme for parallelization of DEBS.

[3]Exceptions to this can occur for nonlinear operations such as $\mathbf{v} \cdot \nabla \mathbf{v}$. Additionally, this advective term uses upwind differences rather than central differences in radial derivatives for stability reasons (see [8]), further complicating this notion.

1.4.3 Semi-Implicit Time Advance

Semi-implicit methods produce stable numerical time advances even for very large time steps. A semi-implicit scheme is developed by subtracting a linear term that mimics one of the non-linear spatial derivative terms from both sides of the given model equation. This leads to numerical behavior similar to implicit methods without the complication and cost of an implicit method when non-linear terms are present. In fact, semi-implicit schemes do not take substantially more time to calculate a single time step compared to explicit methods, but do not have the same stringent stability criteria and thus can lead to a large efficiency gain when longer time scales are of interest in the calculation.

In the case of DEBS solving the MHD equations, long time scale dynamics of the plasma are desired. However, many electrostatic and fast compressional waves arise from the MHD equations. These waves can have short wavelengths and large phase velocities so they will evolve on very short time scales. Explicit time stepping of hyperbolic equations will have numerical instability if the time stepping is too large to resolve wave propagation along the simulation mesh. To have numerical stability in these situations a Courant-Friedrichs-Lewy (CFL) condition must be met. This means that explicit methods are restricted to time stepping

$$\Delta t < \frac{\Delta x}{v} \tag{1.17}$$

where Δx is the mesh spacing and v is related to the wave velocity. Thus the set of fast MHD waves would impose extremely limiting time stepping. For DEBS semi-implicit terms are used to numerically stabilize these waves. These semi-implicit operators are chosen such that they do not couple Fourier components and that only nearest neighbor radial mesh points are coupled. This allows for time stepping in DEBS to be done

with only block tridiagonal matrix solves when these semi-implicit terms are included. Solving tridiagonal matrices requires on the same order of operations as explicit stepping. This allows DEBS to be nearly as fast as an explicit code while having the larger time stepping advantages of strictly implicit codes. While increased time stepping does affect the numerical accuracy of the time-dependent dynamics of these waves, their dynamics are not the main focus of DEBS simulations. It should be noted that the $\mathbf{v} \cdot \nabla \mathbf{v}$ term in equation (1.2) is treated explicitly and does not have a corresponding semi-implicit operation, thus this term still imposes a CFL condition on DEBS. Because the plasma flow speeds are much smaller than the Alfvén speed in RFPs, the CFL condition from advection does not impose a severe limitation.

Chapter 2

Parallelization of DEBS

This chapter will cover the steps taken to produce a parallel DEBS code. The focus of these sections will be primarily to describe the radial domain decomposition. One important goal for the parallelization of DEBS was to minimize the amount of code that needed to be written, while avoiding unnecessary slowdown of the code execution.

2.1 LAPACK

Prior to a complete parallel code, an intermediate step was desired to ensure that successful completion of the DEBS was possible and to reduce the amount of editing that was done in a single step. A substantial part of the parallelization was to replace the built-in, in-place matrix solving routines in DEBS with parallel matrix solving routines from a standard mathematics library. The library identified for this purpose was ScaLAPACK.[9] Thus, a natural stepping stone was the LAPACK library.[10] LAPACK is a serial linear algebra and matrix solving library, which ScaLAPACK is based on. The first step in revising DEBS was then to replace the built-in matrix solver with a LAPACK routine. The LAPACK routine that was chosen was ZGBSV. This routine solves a general banded matrix with double precision complex elements. This solver was chosen since DEBS is required to solve a block tridiagonal system due to the semi-implicit time

stepping method. A block tridiagonal matrix can equally well be described as a banded matrix with a relatively small bandwidth. After converting DEBS to use LAPACK linear algebra routines, the LAPACK solvers were replaced with their ScaLAPACK counterparts. In addition to the ScaLAPACK calls, other direct communication calls were made. This was necessary to update variables beside those advanced in time, such as **B** and **J**.

2.2 OpenBLAS

An early attempt was made to make a simply parallelized version of DEBS by using a shared-memory multi-threaded BLAS library in place of the standard Netlib library. OpenBLAS was chosen for this since it is freely available and supports several modern processor architectures. The OpenBLAS library was able to be linked with the intermediate version of DEBS that included calls to LAPACK. It was found that not only did the multi-threading not improve performance, but actually leads to substantial reductions in speed. Using a two-core processor, a sample calculation using two threads was over three times slower than when using only a single thread.

Investigation of the LAPACK subroutines responsible for solving banded matrices showed that none of the multi-threaded algorithms of OpenBLAS would be used. The multi-threading of OpenBLAS becomes useful when performing matrix-matrix multiplications with large dense matrices. Narrow-banded matrix solving algorithms do not generally involve any of these operations so DEBS was not able to effectively utilize the OpenBLAS library.

2.3 Decomposition with ScaLAPACK

The ScaLAPACK library uses a distributed memory parallel scheme for its matrix solving routines. In an abstract sense this means that individual processors own different unique variables for the set of equations that the matrix represents. For the matrix equations in DEBS, the vector elements that are determined are the fundamental variables at different radial mesh points. Thus, using the ScaLAPACK parallel routines leads to a distribution of the radial mesh. With each radial mesh point comes all of the fundamental variables at that point including the complete set of Fourier components for each of them. One could consider this to be equivalent to dividing up the periodic cylinder into concentric cylinders, which are distributed to each processor. A simpler representation is to only consider the 1-D radial mesh for decomposing the domain, since the semi-implicit operations do not couple Fourier components. For illustration purposes the domain decomposition will be described only using the radial coordinate, with the understanding that each radial point actually represents all the Fourier components of all the variables that exist on it.

There are several properties of the radial mesh as it was designed for DEBS that were carefully considered in the design of this domain decomposition. Firstly, the radial mesh in DEBS is staggered. This means that there is a difference between separating the mesh between r(i) and rp(i) compared to between rp(i) and r(i+1). The other consideration has to do with the legacy code status and age of DEBS. The way DEBS was originally written required that the mesh size be fixed at compile-time as opposed to being dynamically allocated when the program is run. To minimize the amount of changes made to DEBS, a domain decomposition scheme that works around these

considerations was developed.

With array sizes set when the code is compiled, it is natural to define this array size as the radial mesh size per processor. This avoids making major changes to how arrays are handled by DEBS and creates new flexibility since the total mesh size can be changed by adjusting the number of processors for a single compiled DEBS executable. This leads to the choice of having the mesh divided across the boundary between radial points with forms rp(i) and r(i+1), as shown in figure 3. With this domain decom-

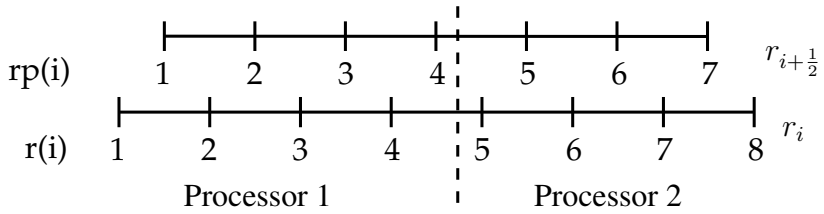

Figure 3: Radial mesh decomposition for parallel DEBS. The radial points from the global mesh that are owned by each processor are separated by the dashed line.

position, the evaluation of radial derivatives next to the processor interface boundary requires explicit communication as a result of the dependencies demonstrated by figure 2 in section 1.4.2. As a result of the finite differences based on the staggered mesh in DEBS, communication is needed in only one direction for evaluating single radial derivatives. The communication scheme for the parallel version of DEBS is shown in figure 4. To accommodate and simplify the programming of evaluation of derivatives following communication calls, the arrays holding variables on the $i+\frac{1}{2}$ were extended to

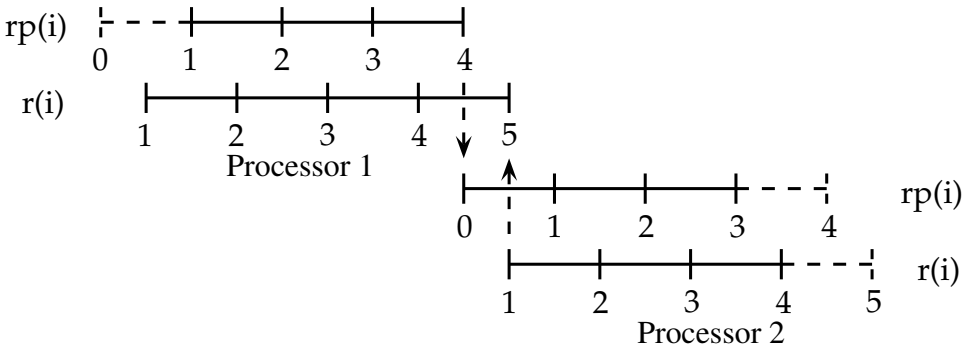

Figure 4: Communication scheme used in parallel DEBS. The points that are indicated by dashed lines are allocated in memory, but go unused in the program. The dashed arrows represent communication of data.

include a 0 index.[1] The greatest array index for variables on the i mesh is also reserved to receive communicated values owned by another processor. Thus, for some mesh size parameter nr, set at compile time, the global mesh scales multiplicatively with the number of processors, such that the global mesh size is defined by nr_tot=(nr−1)∗nprocs. With this scheme for the radial domain decomposition, it is not necessary to edit any of the calculations that are done for the interior of each local domain. An implementation of this domain decomposition is successfully implemented in DEBS without losing any consistency with the original program.

[1]In equation (1.2) both the viscous ($\nabla^2\mathbf{v}$) and advective ($\mathbf{v}\cdot\nabla\mathbf{v}$) terms require treatment beyond this simple scheme. The viscous term includes second derivatives in r, which couples nearest neighbors for each mesh. The advective term does mesh averaging and uses upwind differences so it does not have the convenient information dependency described. To accommodate these problems all velocity arrays are extended to include indices from 0 to nr and extra communication is performed for these variables.

Chapter 3

Results

Various tests have been run to evaluate the performance of the parallel version of DEBS. Execution timing studies of both strong and weak scaling for DEBS are presented here. These calculations are run with various mesh sizes currently used in production level simulations. There is interest in testing cases with much greater resolution in the mesh than has been done previously. In section 3.2 one previously studied physics simulation has been revisited with varied mesh sizes to investigate whether mesh resolution affects these simulations.

3.1 Timing Studies

A range of computational situations were run to evaluate the gain in performance from parallelization of DEBS. These studies provide some insight into the optimal parallel computing configuration for a desired mesh. Two physics test cases are considered these scaling studies. One test case favored a highly resolved radial mesh over the number of spectral modes, while the other test case favored utilizing a large number of spectral modes at the cost of radial resolution. For the purposes of the timing studies, the details of the physics that are being studied is not important. However, it is important to consider mesh resolutions that would realistically be used.

In DEBS there are three parameters that define the size of the computational mesh; nr, mt, and mz. When run in parallel, nr is the size of the radial array allocated to each, and nr_tot is defined in section 2.3 on page 19. Abstractly, the total radial mesh size, nr_tot, will be equivalently referred to as N_r, while nr is related to $\frac{N_r}{P}$, where P is the number of processors. The parameters mt and mz define the number of Fourier components by $N_\theta = 2^{\mathrm{mt}}$ and $N_z = 2^{\mathrm{mz}}$. The size of the meshes used for the poloidal and axial meshes must be a power of two for the FFTs in DEBS.

The computer resources for the timing studies presented here are provided by the UW-Madison Center For High Throughput Computing (CHTC). Their high performance computing (HPC) cluster is used. The HPC cluster compute nodes each contain two Intel Xeon E5-2670v2 processors. This gives each node a total of 20 processor cores. Each node also has 128 GB of RAM. The HPC cluster is connected by a 56 Gbit/s Infiniband network. These nodes were connected to a gluster file storage system, which provided uniform file access across all HPC nodes. To compile the parallel DEBS OpenMPI version 1.6.4 is used. It is also the compiler used for the ScaLAPACK and HDF5 libraries that the parallel DEBS has as dependencies. The ATLAS version of the LAPACK and BLAS libraries is linked, as these provide the best performance.

3.1.1 Strong Scaling

To asses the parallel implementation's ability to reduce run-time, two sets of strong scaling computations were run with different global mesh sizes. The two mesh sizes studied were $N_r = 1000$ and $N_r = 200$, with both having mt = 3 and mz = 5. The results of these runs are summarized in figure 5. The single processor runs of the parallel

version of DEBS are used as the baseline for calculating efficiency. The original serial

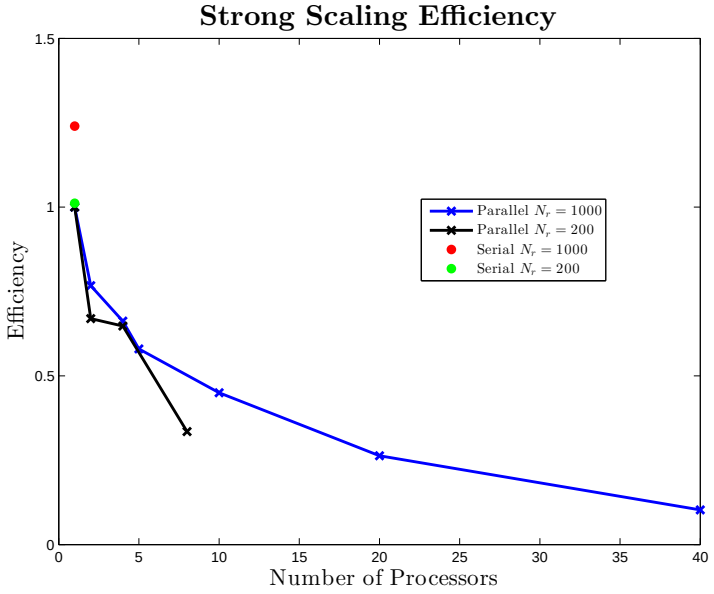

Figure 5: Graph of strong scaling efficiency. Both sets of runs used mt = 3 and mz = 5.

version of DEBS is also compared with the new parallel version. Due to the more efficient single processor algorithm, the original serial DEBS has higher single processor efficiency. However, the speedup gained by just two processors is enough to overcome the difference in execution time between the serial and parallel codes. For example, the serial DEBS and two processor runs for the $N_r = 200$ runs shown in figure 5 had execution times of 22 minutes 46 seconds and 17 minutes 12 seconds, respectively.

With perfect strong scaling, the efficiency would appear as a horizontal line at one. Parallel DEBS does not get close to satisfying this criterion even though decreases in execution time are observed. As can be seen in figure 5, with a smaller radial mesh there is a steeper cost in efficiency as a greater number of processors is used. This behavior is not surprising since there is less and less overlap between the communication calls made

by ScaLAPACK and the numerical matrix solving operations as the ratio $\frac{N_r}{P}$ decreases. With both of these test cases it was found that overall program speedup levels off or even decreases for $\frac{N_r}{P} < 50$. Thus, most cases run with the parallel DEBS would run in the least amount of wall-clock time using this value. However, the utilization of CPU resources is low by this point.

3.1.2 Weak Scaling

For the weak scaling study one set of Fourier components was used for the sake of simplicity in presenting the data. The values mt = 3 and mz = 5 are set for these runs. Four sets of $\frac{N_r}{P}$ values are used to investigate the weak scaling in relation to that ratio. The results of these studies are displayed in figure 6. It should be noted that for the single processor case, the execution time scaled linearly with N_r. This is expected as banded matrix solving algorithms based on Gaussian elimination use $O(N_r)$ operations. Likewise, the number of FFT solves scales directly with N_r since there are the same number of Fourier components for each radial mesh point. This confirms that these calculations are done in a regime where the matrix size is much greater than the band width. The total bandwidth for the semi-implicit matrix representation of equation (1.1) is 11, whereas the matrix size is $3N_r$.

These weak scaling study data are shown in a semi-log plot of execution time versus number of processors. The number of processors is displayed logarithmically. From this plot is appears that the execution time scales linearly with the logarithm of the number of processors up until about 20 processors. Beyond this point there is a steeper increase in execution time. The 20 processor value is significant as that is the number of CPU cores

on each compute node. Since execution time increases substantially with the number of processors in each series of calculations, there must be a significant communication cost associated with the parallel algorithms used by DEBS. However, the amount of explicit communication added outside of the calls to ScaLAPACK should scale linearly with the number of processors. This does not fit the trends shown in figure 6, thus an explanation associated with ScaLAPACK was investigated.

The exact ScaLAPACK banded matrix solver that is called by the parallel DEBS is PZDBSV. This assumes a diagonally dominant-like matrix, thus does not use any pivoting. Using this algorithm has not shown any difference in the solutions of DEBS aside from some minor round-off errors due to slightly different orders of operations when compared to the serial version of DEBS. In fact, the matrix solving algorithm used in the serial version of DEBS also depends on the diagonally dominant nature of the

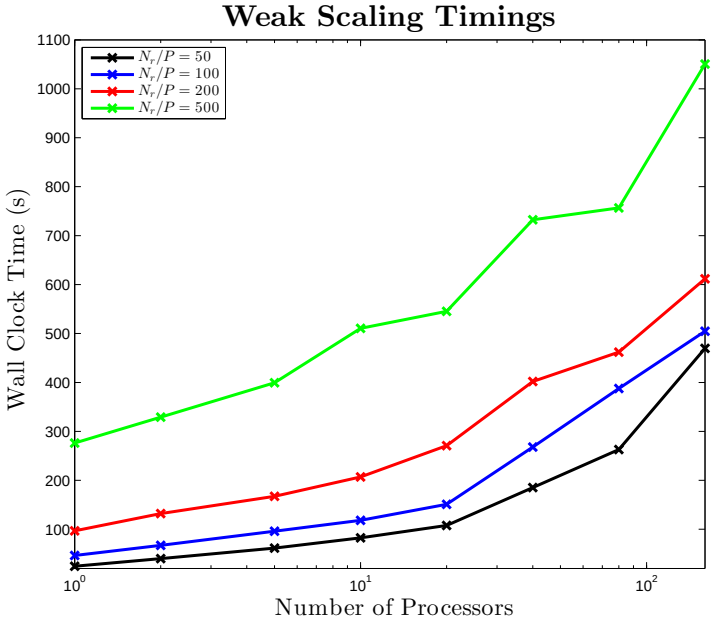

Figure 6: Graph of weak scaling execution time. All runs used mt=3 and mz=5 and were advanced for 1000 time steps.

underlying system. The banded matrix solving scheme used in the PZDBSV subroutine is a divide-and-conquer algorithm.[11, 12] This algorithm reorders the input matrix to generate mostly independent matrix equations on each processor. This reordering has communication costs that scale with the logarithm of the number of processors. More specifically, the total operation count of the ScaLAPACK routine in question is [12]

$$\begin{aligned}\varphi \approx\ & 2k_l(4k_u+1)\frac{n}{p} + (4k_l+4k_u+1)r\frac{n}{p} \\ & + \left(\frac{32}{3}k^3 + 9k^2r + 6t_s + 4k(k+r)t_w\right)\lfloor \log_2(p-1)\rfloor.\end{aligned} \tag{3.1}$$

Here p is the number of processors, n is the matrix size, r is the number of right hand sides, k_l is the number of lower matrix bands, k_u is the number of upper bands, and $k = \max(k_l, k_u)$. The other variables, t_s and t_w, are related to the cost of communication. The symbol t_s represents the start-up time for a single communication call in terms of floating point operations (flops). In a similar fashion, t_w is a measure of the time needed to transmit a word of data. In this instance a word is a (16-Byte) complex floating point number. The actual values of t_s and t_w can vary significantly depending on the exact architecture and setup of the computer system being used. As of yet, no benchmarks have been done to measure t_s and t_w on the computer system used for the studies presented here, but Arbenz et al. [12] suggests that reasonable estimates in this case are $t_s \approx 1000$ and $t_w \approx 10$. Within DEBS three different sized matrices are solved with values $k_l = k_u = k = 1, 3, 5$ and $n = N_r, 2N_r, 3N_r$. In order to find the dominating operation count terms we will substitute $r = 1$, $k_l = k_u = k = 1$, and $n = N_r$ into equation (3.1). This choice emphasizes the communication terms the most, which are hypothesized to give the limiting behavior seen in the weak scaling results. This gives a

simplified operation count.

$$\varphi \approx 19\frac{N_r}{p} + (20 + 6t_s + 8t_w)\lfloor \log_2(p-1) \rfloor \tag{3.2}$$

In this case the cost of starting each communication call during matrix reordering tends to dominate the total operation count for the majority of the $\frac{N_r}{P}$ values studied, even with low numbers of processors.

With the analysis that has been presented thus far, it has been established that within the regime of mesh sizes that would reasonably be used for DEBS the execution time for weak scaling should increase linearly with the logarithm of the number of processors with a slope related to t_s. However, figure 6 suggests that there are two distinct values of t_s over different ranges of P. As previously pointed out, the transition between these two slopes occurs exactly at the number of processors present on a single compute node for these tests. From this it can be inferred that communication between processors within a single piece of hardware is much faster than the communication through the inter-connect hardware between separate compute nodes. Communication between computers must be done over some sort of network, whereas exchanging data between processors on a single compute node only involves moving data around on the computer memory, i.e. RAM. Sending data across a network has much greater latency and less bandwidth than there is for RAM. Since the computing scenarios studied for the weak scaling of DEBS appear to be limited by the communication start-up time, which depends on a hand-shaking procedure between processors that is dependent on message latency, DEBS calculations with modest values of $\frac{N_r}{P}$ should be executed mostly on a single piece of hardware to minimize latency of communications.

3.2 Physics Test Case

To exercise the parallel implementation on production level calculations, a case previously studied by Dr. Josh Reusch is revisited. These calculations are long time scale studies of sawtooth oscillations. Due to time constraints only a very small portion of the original simulation time was repeated. Even with this limitation, at least one complete sawtooth cycle has been simulated in three computations. The meshes set for these calculations are $N_r = 200$ mt $= 4$ mz $= 8$, $N_r = 500$ mt $= 4$ mz $= 8$, and $N_r = 500$ mt $= 3$ mz $= 6$. For each of these cases $\frac{N_r}{P} = 50$, thus four, eight, and ten processors are used, respectively. The approximate wall-clock times taken to advance each of these cases the same amount of simulation time are 2 days 5 hours 40 minutes, 5 days 1 hour 29 minutes, and 9 hours 15 minutes, respectively. For these simulations the Lundquist number is set to 10^6, magnetic Prandtl number is set to 100, and a maximum cell Reynolds number is set to 0.1. In order to satisfy the maximum cell Reynolds number, artificial viscosity is computed at each time-step, based on the current flow field. This is used for numerical stability reasons. The output from simulations done for these mesh sizes are collected and the results are presented here.

In figure 7 the reversal parameter is plotted over the simulation time. As can be seen in this plot, there are several substantial differences in the outcomes. Between the traces with $N_r = 200$ and $N_r = 400$ there appears to be almost a factor of two difference in the time period over which this single sawtooth cycle occurs. This observation shows the sensitivity of the nonlinear RFP simulations to numerical resolution.

During a sawtooth crash there exists strong nonlinear behavior of the plasma. This can excite large local flow fluctuations, representing turbulent cascading of energy to the

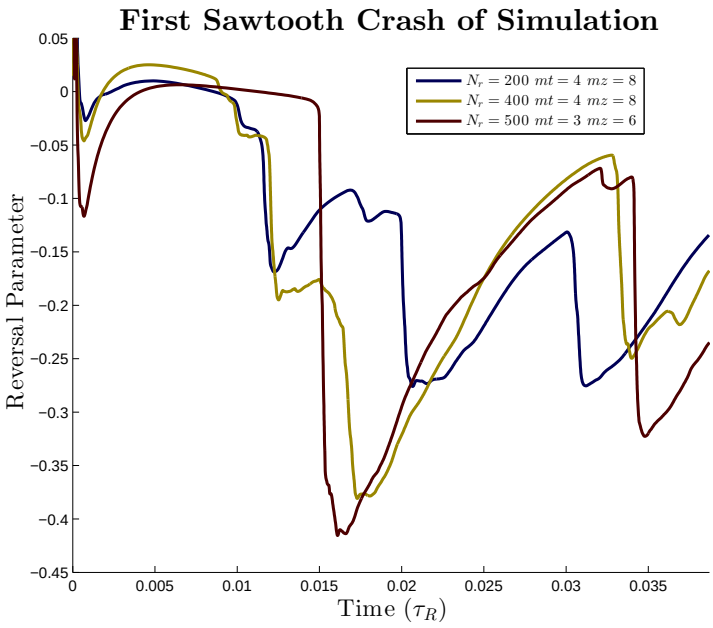

Figure 7: Simulations with equivalent initial conditions were run for several mesh sizes. Enough simulation time was advanced to observe a full sawtooth oscillation.

dissipation length scale. In order to prevent nonlinear numerical stability at small values of physical viscosity during these situations, the artificial viscosity in DEBS damps out these perturbations on the mesh scale for a given N_r value. This is allowable since these particular dynamics do not necessarily need to be accurately modeled in order to accurately model the long time scale plasma dynamics of interest. However, it was hypothesized that increased radial resolution may reduce the need or at least the magnitude of the artificial viscosity that would be introduced. Using the same simulation as in figure 7, the non-dimensional viscosity, i.e. magnetic Prandtl number, is displayed in figure 8. From this plot it seems that for the simulations studied, there is not a decrease in the amount of artificial viscosity that must be used.

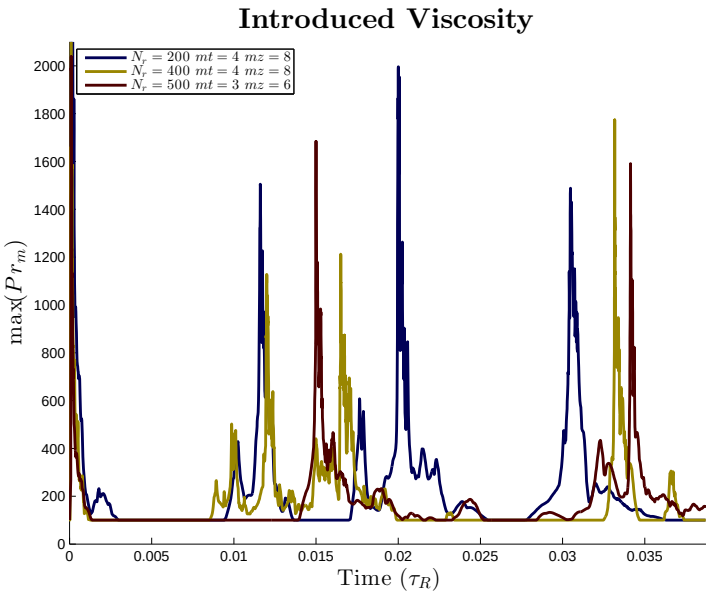

Figure 8: For each of the cases presented in figure 7, the viscosity is plotted here. During each sawtooth crash a short period of large viscosity is introduced.

Chapter 4

Conclusions

The DEBS code was successfully parallelized using calls to the ScaLAPACK library in combination with additional MPI programming. The radial domain decomposition scheme that was designed for DEBS was found to be effective and did not require rewriting any substantial portion of the physics computations. The resulting parallel DEBS was found to produce results that are numerically consistent with the serial version of DEBS used as a starting point. This was important and necessary since none of the underlying physics of DEBS was changed.[1]

Overall performance of the newly finished parallel DEBS code is evaluated with respect to both strong and weak scaling studies. From the strong scaling tests it is found that parallel execution of the code starts to become ineffective with $\frac{N_r}{P} < 50$. In future production calculations this value should be used as a lower bound when deciding how many processors to use for a calculation with a desired N_r. With the weak scaling tests at radial mesh resolutions relevant to RFP simulation, the limiting factor of parallel

[1] A bug was found in the calculation of $\mathbf{v} \cdot \nabla \mathbf{v}$. A fix for this bug was sent out to DEBS users prior to the completion of the parallel code and was included in the changes. This fix did produce a numerical difference in the DEBS output. However, there did not seem to be a qualitative difference in the output, although this was not extensively investigated. One piece of the bug was an indexing error, which reduced the order of accuracy of DEBS. There was also a factor of $\frac{1}{r}$ missing from one of the terms of $\mathbf{v} \cdot \nabla \mathbf{v}$, which in rare cases might lead to more substantial error.

execution time depends on the latency associated with the initiation of inter-processor communications. For this reason, production level simulations are best performed within a shared-memory unit of hardware. This is because the latency of communications across a network is much greater than within a compute node. For very large radial mesh resolution this condition could be relaxed. The timing studies presented in this thesis utilize the OpenMPI implementation of the MPI standard. Additional investigation could be done with several other MPI implementations to determine if some other implementation has lower communication start-up costs, and would thus be more efficient for use with DEBS.

A single physics test case for DEBS investigates numerical resolution over a limited range of computational mesh sizes. With a numerically converged simulation, increasing mesh resolution does not significantly change the numerical simulation results beyond some desired numerical tolerance. The $N_r = 200$ test case as seen in figure 7 represents a somewhat practical limit of the mesh that can be reasonably calculated with the serial DEBS code. From that figure it is quite clear that numerical convergence of the DEBS mesh has not been reached. It is possible that even without numerical convergence, the cycle averaged characteristics of sawteeth would not be very sensitive to computational mesh size. The initial results presented here do provide some evidence to believe that even cycle averaged sawtooth behavior may change with mesh resolution. This is supported by the substantial differences in cycle time for the single simulated sawtooth. Only simulations with many more sawtooth oscillations will be able to determine if averaged sawtooth behavior is indeed changed with greater radial resolution. With the new parallel DEBS, improved simulation turnaround time will allow this question, and others, to be investigated. General production computations will also now be able to be

performed with greater accuracy within similar time frames.

Even with the improvements brought by the parallel DEBS, there are still some limitations to the code. As mentioned above, there are only certain computational mesh configurations that are truly able to leverage the power of parallel computing for calculations of interest with DEBS. After parallelization, there is no longer as much limitation on the time it takes to calculate a single time step. Now the limiting factor is mostly the number of time steps that must be calculated to advance the simulation time through a desired point. This becomes increasingly important as the radial mesh is further resolved, since this imposes a CFL condition due to the advancement of the advective flows. With extremely large radial mesh sizes that may be needed to numerically converge some DEBS simulations, numerical stability with the explicit advance of advection may require time steps that are much smaller than is necessary to observe the desired dynamics. Even more limiting may be that these calculations may be too impractical, despite parallel improvements, due to the number of time steps they would require. This leaves room for future work in stabilizing the advective flow term of the MHD equations as solved by DEBS. While it is not possible to numerically stabilize the advective flow term with semi-implicit operations, an implicit method may be used.[13] However, this would require a complete 3-D inversion of the velocity equations, since an implicit method would couple Fourier components. This is counter to the philosophy behind the DEBS code so it is an unlikely change.

Despite the existence of some expected limitations, the improvements made through parallelization of DEBS will open the door for previously untested calculations. Along with these brand new simulations, it may be worth revisiting some of the older findings of DEBS in order to confirm those results. The new parallel DEBS is expected to become

an important tool for the MST group and others to produce simulations relevant to their experimental findings.

Appendix A

References

[1] D. D. Schnack, D. C. Barnes, Z. Mikic, D. S. Harned, and E. J. Caramana, "Semi-implicit magnetohydrodynamic calculations", Journal of Computational Physics **70**, 330 (1987).

[2] S. C. Prager, A. F. Almagri, S. Assadi, J. A. Beckstead, R. N. Dexter, D. J. Denhartog, G. Chartas, S. A. Hokin, T. W. Lovell, T. D. Rempel, J. S. Sarff, W. Shen, C. W. Spragins, and J. C. Sprott, "1st results from the Madison Symmetric Torus reversed field pinch", Physics of Fluids B-Plasma Physics **2**, 1367 (1990).

[3] R. N. Dexter, D. W. Kerst, T. W. Lovell, S. C. Prager, and J. C. Sprott, "The Madison Symmetrical Torus", Fusion Technology **19**, 131 (1991).

[4] J. A. Reusch, "Measured and simulated electron thermal transport in the Madisom Symmetric Torus reversed field pinch", PhD thesis (The University of Wisconsin - Madison, 2011).

[5] D. W. Walker, "The design of a standard message-passing interface for distributed-memory concurrent computers", Parallel Computing **20**, 657 (1994).

[6] J. P. Boyd, *Chebyshev and Fourier spectral methods* (Courier Dover Publications, 2001).

[7] J. W. Cooley and J. W. Tukey, "An algorithm for machine calculation of complex Fourier series", Mathematics of Computation **19**, 297 (1965).

[8] R. Lionello, Z. Mikic, and J. A. Linker, "Stability of algorithms for waves with large flows", Journal of Computational Physics **152**, 346 (1999).

[9] L. S. Blackford, S. for Industrial, and A. Mathematics., *ScaLAPACK user's guide*, Software, environments, tools (SIAM, Philadelphia, 1997), xxvi, 325 p.

[10] E. Anderson, *LAPACK users' guide* (Society for Industrial and Applied Mathematics, Philadelphia, 1992), xv, 235 p.

[11] A. Cleary and J. Dongarra, "Implementation in ScaLAPACK of divide-and-conquer algorithms for banded and tridiagonal linear systems", Computer Science Dept. Technical Report CS-97-358, University of Tennessee, Knoxville, TN (1997).

[12] P. Arbenz, A. Cleary, J. Dongarra, and M. Hegland, "A comparison of parallel solvers for diagonally dominant and general narrow-banded linear systems", Eidgenssische Technische Hochschule, Department of Computer Science, Institute of Scientific Computing (1998).

[13] C. R. Sovinec and J. R. King, "Analysis of a mixed semi-implicit/implicit algorithm for low-frequency two-fluid plasma modeling", Journal of Computational Physics **229**, 5803 (2010).