# Parallelization of a Particle-in-Cell Simulation Modeling Hall-Effect Thrusters

by

Justin M. Fox
B.Sc. California Institute of Technology (2003)

Submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2005
[February 2005]

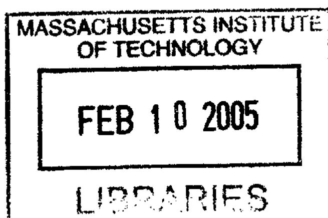
© Massachusetts Institute of Technology 2005, All rights reserved.

Author ..........................................................
Department of Aeronautics and Astronautics
January 14, 2005

Certified by ..........................................................
Manuel Martinez-Sanchez
Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by ..........................................................
Oleg Batishchev
Principal Research Scientist
Thesis Supervisor

Accepted by ..........................................................
Jaime Peraire
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Studies


MASSACHUSETTS INSTITUTE OF TECHNOLOGY
FEB 1 0 2005
LIBRARIES

AERO

# Parallelization of a Particle-in-Cell Simulation Modeling Hall-Effect Thrusters

by

Justin M. Fox

Submitted to the Department of Aeronautical and Astronautical Engineering
on January 14, 2005, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautical and Astronautical Engineering

## Abstract

MIT's fully kinetic particle-in-cell Hall thruster simulation is adapted for use on parallel clusters of computers. Significant computational savings are thus realized with a predicted linear speed up efficiency for certain large-scale simulations. The MIT PIC code is further enhanced and updated with the accuracy of the potential solver, in particular, investigated in detail. With parallelization complete, the simulation is used for two novel investigations. The first examines the effect of the Hall parameter profile on simulation results. It is concluded that a constant Hall parameter throughout the entire simulation region does not fully capture the correct physics. In fact, it is found empirically that a Hall parameter structure which is instead peaked in the region of the acceleration chamber obtains much better agreement with experiment. These changes are incorporated into the evolving MIT PIC simulation. The second investigation involves the simulation of a high power, central-cathode thruster currently under development. This thruster presents a unique opportunity to study the efficiency of parallelization on a large scale, high power thruster. Through use of this thruster, we also gain the ability to explicitly simulate the cathode since the thruster was designed with an axial cathode configuration.

Thesis Supervisor: Manuel Martinez-Sanchez
Title: Professor of Aeronautics and Astronautics

Thesis Supervisor: Oleg Batishchev
Title: Principal Research Scientist

# Acknowledgments

There are so many people who have helped make this possible that I could never possibly thank everyone in just one page. Of course, I should single out Professor Martinez-Sanchez and the MIT Aero/Astro Department for the opportunity to conduct this research. Profound thanks also to Dr. Oleg Batishchev for his invaluable help and guidance along the way. To both of my advisors, you have my utmost gratitude and respect for all the help you have given, to me and the others. Thank you to Shannon and Jorge, my adopted big sister and big brother in the lab. Your kind words, advice, food, and laughter meant a great deal more to this scared little kid than you probably realized. Sincere thanks also to Kay for taking so much time away from her own work to help me understand why there were aardvarks in the code. I couldn't have gotten anywhere without your experience and kindness.

Outside MIT, I must acknowledge Dr. James Szabo for creating the original MIT PIC code and thank him and his company Busek for all of their help and patience along the way. Thanks also to Mr. Kamal Joffrey, PE and his colleagues at DeltaSearch Labs for making my time there pleasant, profitable, and enjoyable.

Of a more personal nature, I would like to recognize the incredible impact my once-mentor Dr. Rebecca Castano has had on my life, and thank her once again for introducing me to the joys of research. To my brother Jason, thank you for blazing the trail and forever staying one step ahead, forcing me to run that much harder. It has always been an honor to be your brother. Of course, the deepest thanks and love to my fairy tale princess, my light in the darkness, my best friend in the whole world, Corinne. It is your shoulder I lean most upon, but your unflagging optimism and buoyant spirit have never let me fall. And finally and most importantly, thank you to my mother and my father for...well...everything. I hope this was worth all those nights reading me *Green Eggs and Ham* and letting me win at *Memory Match*. ☺

I'd also like to thank my High School friends, my fourth grade teacher, my cat...

# Contents


## 1 Introduction

1.1 Background of Hall-Effect Thrusters . . . . . . . . . . 20

1.2 The P5 Thruster . . . . . . . . . . 22

1.3 Previous Hall Thruster Simulation Work . . . . . . . . . . 23

1.4 Full-PIC Versus Other Methods . . . . . . . . . . 24

1.5 Physical Acceleration "Tricks" . . . . . . . . . . 25

1.6 Introduction to Parallelization . . . . . . . . . . 26

1.7 Thesis Objectives . . . . . . . . . . 28

1.8 Organization of the Thesis . . . . . . . . . . 28

## 2 The Serial Code

2.1 Summary of Simulation Technique . . . . . . . . . . 31

2.2 Structure of the Serial Simulation . . . . . . . . . . 31

2.3 Initialization . . . . . . . . . . 32

2.4 Electric Potential Calculation . . . . . . . . . . 34

2.5 Particle Movement . . . . . . . . . . 35

2.6 Modeled Collisions . . . . . . . . . . 37

2.7 Data Collection . . . . . . . . . . 37

## 3 Parallelization of Electric Potential Calculation

3.1 Resources Used . . . . . . . . . . 40

3.2 Introduction to MPI and MPICH . . . . . . . . . . 40

3.3 Gauss's Law Solver . . . . . . . . . . 42

3.4 Parallelization of Successive Over-Relaxation . . . . . . . . . . 45

3.5 Implementation of Gauss's Law Parallelization . . . . . . . . . . 49

3.6 Preliminary Results for Gauss's Law Parallelization . . . . . . . . . . 52

3.7 Ground-Truthing of the Gauss's Law Solver . . . . . . . . . . 54

3.8 Further Ground-Truthing of the Gauss's Law Solver . . . . . . . . . . 58

3.9 Convergence of Successive Over-Relaxation . . . . . . . . . . 60

3.10 Speed up of the Parallel Solver . . . . . . . . . . 62

3.11 Speed up of the Parallel Solver on a Larger Grid . . . . . . . . . . 67

## 4 Parallelization of Particle Mover and Collisions

4.1 Parallelization of Particle Distribution Creation . . . . . . . . . . 70

4.2 Parallelization of Neutral Injection . . . . . . . . . . 72

4.3 Parallelization of Ion-Neutral Charge Exchange . . . . . . . . . . 73

4.4 Parallelization of Ion-Neutral Scattering . . . . . . . . . . 75

4.5 Parallelization of Neutral Ionization . . . . . . . . . . 76

4.6 Parallelization of Double Ionization . . . . . . . . . . 77

4.7 Parallelization of Cathode Electron Injection . . . . . . . . . . 78

4.8 Preliminary Speed up of Particle Mover . . . . . . . . . . 79

## 5 Overall Parallelization Results


5.1 Preliminary Results of the Fully-Parallelized Code .............. 86

5.2 Preliminary Timing Results .............................. 86

5.3 Analysis of Electron Number Data ........................... 87

5.4 Analysis of Neutral Number Data ........................... 88

5.5 Analysis of Anode Current Results ........................... 90

5.6 Fixing Non-Convergence .................................. 91

5.7 Addition of Chebyshev Acceleration to SOR .................. 97


## 6 Investigations Using the Parallel Code


6.1 Implementation of a Variable Hall Parameter .................. 102

6.1.1 Structure of Varying Hall Parameter .................... 102

6.1.2 Results Comparing Varying and Constant Hall Parameter ... 103

6.1.3 Comparing Different Hall Parameter Structures ........... 108

6.2 Modeling of a High Power Central Cathode Thruster ............. 111

6.2.1 The Central Cathode Thruster and Simulation Grid ........ 111

6.2.2 High Power Thruster Magnetic Field Structure ........... 113

6.2.3 Results of High Power Thruster Modeling ............... 115

6.2.4 Discussion of the High Power Thruster and Results ........ 120

**7 Conclusions and Future Work**

7.1 Summary of Work . . . . . . . . . . 123

7.2 Possible Future Work . . . . . . . . . . 125

**A Useful MPICH Functions**

# List of Figures


1-1 Schematic Representation of a Hall Thruster . . . . . . . . . . . . . . . . . . . . . . . . 21

1-2 The P5 Thruster in (a) a schematic representation and (b) as a computational grid structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

2-1 A Flowchart of the Overall Serial Code Structure . . . . . . . . . . . . . . . . . . . . . . 32

2-2 Magnetic field in the chamber of the P5 thruster . . . . . . . . . . . . . . . . . . . . . . 34

3-1 The Serial Nature of SOR. To calculate $\Phi^{(k+1)}$ at point (i,j), we need to have already calculated $\Phi^{(k+1)}$ below and to the left of (i,j) . . . . . . . . . . . . . . . . . . 46

3-2 Red-Black SOR Algorithm with ovals representing black nodes and rectangles representing red. (a) The $k^{th}$ iteration begins. (b) The black nodes update first using the red values from the $k^{th}$ iteration. (c) The red nodes update using the black values from the $(k+1)^{th}$ iteration. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

3-3 Parallelizing SOR. (a) Each processor receives a strip of nodes. (b) The top row of nodes is calculated first. These are the "black" nodes. (c) The remaining "red" nodes are then calculated from top to bottom first, then from left to right . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48

3-4 Average Electric Potential of (a) Serial Code and (b) Parallel Code Over Similar Experiments. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53

3-5 Averaged Electron Temperatures of (a) the serial code and (b) the parallel code. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54

3-6 The Analytic (a) potential and (b) charge used in the ground truth tests. . . . . . 55

3-7 The normalized magnitude of the difference between the analytic and calculated potentials was found for (a) 1 (b) 2 (c) 4 (d) 8 (e) 16 (f) 32 processors. The calculated potentials agreed to the ninth decimal place. . . . . . . . . . . . . . . . . . . 57

3-8 The normalized difference between the calculated and analytic potentials for the Gauss's Law Solver running on (a) 1 (b) 2 (c) 4 (d) 8 (e) 16 and (f) 32 processors. All plots are on the same scale. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59

3-9 Plots of the residue RHS vs. over-relaxation iteration for (a) 1 (b) 2 (c) 4 (d) 8 (e) 16 (f) 32 processors. Convergence is exponential, but slightly slower for larger numbers of processors. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

3-10 Time spent in (a) Gauss Function and (b) total iteration for various numbers of processors. These data give credence to the belief that the Gauss function is requiring the greatest amount of total iteration time. Note the peaks at iterations 9 and 19. These occur because every 10 iterations, moments are calculated and data is saved. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63

3-11 Percentages of serial computation time spent in (a) Gauss function and a (b) total iteration. Adding processor after about 16 does not appear to obtain a significant savings. This may not be the case on a larger problem. . . . . . . . . . . . . . . . . 65

3-12 Speed ups of (a) Gauss function alone and (b) the total simulation iteration. Clearly, the algorithm does not achieve the optimal linear speed up ratio. This is likely due to the decreasing rate of convergence of the SOR algorithm as processors are added. Also, as processors are added, the broadcast communication time may become significant. Overall, however, there is a significant reduction in the time required for a simulation iteration. . . . . . . . . . 66

3-13 Speed up of the Gauss's Law Function on a ten times finer grid. . . . . . . . . . . . 68

4-1 The total time for a single simulation iteration was plotted over the first 20 simulation iterations for the cases of 1, 2, 4, 8, and 16 processors. Figure 4-1(a) shows the times for a simulation having on the order of 100,000 particles while Figure 4-1(b) shows the times for a simulation having on the order of 1,000,000 particles. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81

4-2 The time required to do everything except calculate the potential was plotted in the figures above for the cases of 1, 2, 4, 8, and 16 processors. Figure 4-2(a) shows data from a simulation containing on the order of 100,000 particles and Figure 4-2(b) shows data from a simulation containing on the order of 1.000,000 particles. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82

4-3 The percentage of serial time required to move (a) 100,000 particles and (b) 1,000,000 particles was plotted for the cases of 2, 4, 8, and 16 processors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83

4-4 The speed up of the particle mover portion of the simulation was plotted for (a) a simulation of approximately 100,000 particles and (b) a simulation of approximately 1,000,000 particles. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 84

5-1 Number of electrons versus simulation iteration for various parallel cases . . . . 88

5-2 Number of neutrals versus simulation iteration for various parallel cases . . . . . 89

5-3 Anode Current Versus simulation iteration for various parallel cases . . . . . . . . 90

5-4 The absolute differences between the (a) electric potential, (b) Z electric field, and (c) R electric field for a converged SOR iteration versus a non-converged iteration. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93

5-5 Depicting the electron number versus normalized simulation time (1T ~ 7E-9s) for three different serial experiments with varying Gauss's Law over-relaxation parameters . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95

5-6 Depicting the electron number versus normalized simulation time (1T ~ 7E-9s) for two different parallel experiments with varying Gauss's Law over-relaxation parameters . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95

5-7 The convergence rate of SOR without Chebyshev acceleration is shown for (a) 2 processors and (b) 4 processors. The convergence rate for SOR with Chebyshev acceleration added is then shown for (c) 2 processors and (d) 4 processors. . . 100

6-1 (a) The experimentally-measured Hall Parameter variation for the Stanford Hall Thruster at three different voltages and (b) the estimated Hall Parameter variation for the P5 thruster at 500 Volts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 103

6-2 The time-averaged electron energy (eV) for (a) experimental results from the University of Michigan, (b) the MIT PIC simulation with constant Hall parameter, and (c) the MIT PIC simulation with a variable Hall parameter . . . . . . . . . . . 105

6-3 The time-averaged electric potential for (a) the experimental University of Michigan data, (b) the MIT PIC simulation with constant Hall parameter, and (c) the MIT PIC simulation with variable Hall parameter . . . . . . . . . . . . . . . . . . . 107

6-4 Three Hall parameter profiles studied. Each is similar to that used in section 6.1.2, except that (a) has a higher peak, (b) has been shifted left, and (c) has a narrow peak . . . . . . 108

6-5 Electron temperature and potential for the three different Hall parameter profiles: (a)(b) the higher peak, (c)(d) the left-shifted peak, (e)(f) and the narrow peak . . . . . . 110

6-6 Gridding of the high power central-cathode thruster. (a) Originally an indentation was left near the centerline for the cathode, but (b) the final grid does not explicitly model the near-cathode region . . . . . . 112

6-7 The magnetic field structure of the high power central-cathode thruster . . . . . 113

6-8 Time-averaged electric potential for the high power thruster . . . . . . 116

6-9 Time-averaged neutral density ($cm^{-3}$) for the high power thruster . . . . . . 117

6-10 Time-averaged ion temperature (eV) for the high power thruster . . . . . . 118

6-11 Time-averaged electron temperature (eV) for the high power thruster . . . . . . 119

# List of Tables

2-1 Parameters used in all trials unless otherwise noted . . . . . . . . . . . . . . . . . . 33

3-1 Summary of the major communications for the Gauss's Law solver. . . . . 52

6-1 Performance characteristics for the P5 thruster . . . . . . . . . . . . . . . . . . . . . 104

# Chapter 1

# Introduction

## 1.1 Background of Hall-Effect Thrusters

Experimentation with Hall-effect thrusters began independently in both the United States and the former Soviet Union during the 1960's. While American attentions quickly reverted to seemingly more efficient ion engine designs, Russian scientists continued to struggle with Hall thruster technology, advancing it to flight-ready status by 1971 when the first Hall thruster space test was conducted. With the relatively recent increase in communication between the Russian and American scientific communities, American interest has once more been piqued by the concept of a non-gridded ion-accelerating thruster touting specific impulses in the range of efficient operation for typical commercial North-South-station-keeping (NSSK) missions.

A schematic diagram of a typical Hall thruster is shown in Figure 1-1. The basic configuration involves an axially symmetric hollow channel lined with a ceramic material and centered on a typically iron electromagnetic core. Surrounding the channel are more iron electromagnets configured in such a way as to produce a more or less radial magnetic field. A (generally hollow) cathode is attached outside the thruster producing electrons through thermionic emission into a small fraction of the propellant gas. A portion of the electrons created in this way accelerate along the Electric potential toward the thruster channel where they collide with and ionize the neutral gas atoms being

emitted near the anode plate. Trapped electrons then feel an $E \times B$ force due to the applied axial electric field and the radial magnetic field, causing them to drift azimuthally in the manner shown, increasing their residence time. The ions, on the other hand, have a much greater mass and therefore much larger gyration radius, and so are not strongly affected by the magnetic field. The major portion of these ions are then accelerated out of the channel at high velocities producing thrust. Farther downstream, a fraction of the remaining cathode electrons are used to neutralize the positive ion beam which emerges from the thruster.

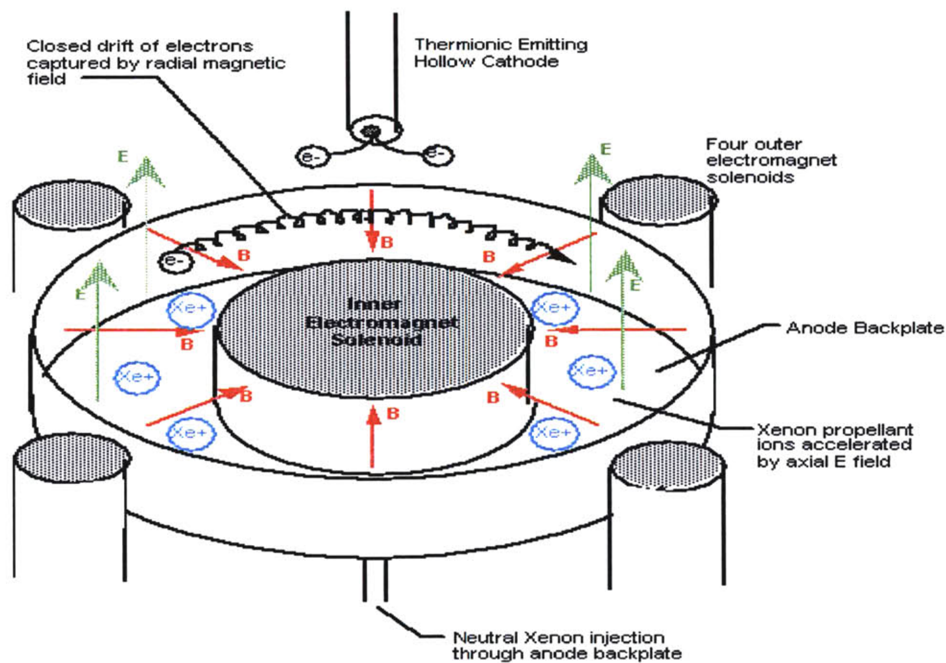



**Figure 1-1: Schematic representation of a Hall thruster**

The Xenon ion exit velocity of a Hall thruster can easily exceed 20,000 m/s, translating into an $I_{sp}$ of over 2000s. This high impulse to mass ratio is characteristic of electric propulsion devices and can be interpreted as a significant savings in propellant

mass necessary for certain missions. In addition to this beneficial trait, Hall thrusters can be relatively simply and reliably engineered. Unlike ion engines, there is no accelerating grid to be eroded by ion sputtering, making Hall thrusters better-suited for missions requiring long-life engines. Finally, Hall-effect thrusters are capable of delivering an order of magnitude higher thrust density than ion engines and can therefore be made much smaller for comparable missions.

## 1.2 The P5 Thruster

The PIC (particle-in-cell) code as given to us by Kay Sullivan [25] was in particular configured to model the P5 Hall-effect thruster. This 3kW SPT (Stationary Plasma Thruster) was developed by Frank Gulczinski at the University of Michigan. The thruster is typically operated at 300 Volts, a current of 10 Amps, and a Xenon gas mass flow of 11.50 mg/s. Whenever possible, these parameters were held constant throughout our simulations. A schematic representation of the radial cross-section of the P5 thruster is shown in Figure 1-2(a) while the typical computational grid used in our numerical simulation is depicted in Figure 1-2(b). For more detailed specifications and design parameters of the P5 thruster, see, for example reference [12].

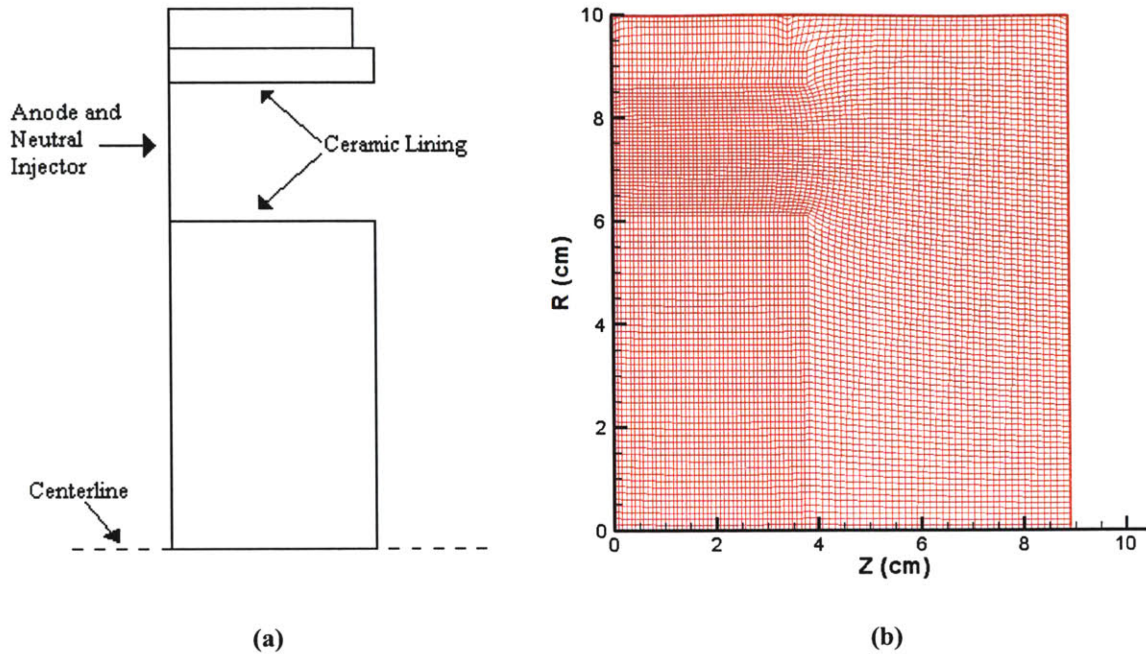



**Figure 1-2: The P5 thruster in (a) a schematic representation and (b) as a computational grid structure.**

## 1.3 Previous Hall Thruster Simulation Work

Due to both their recent increase in popularity and also a lack of complete theoretical understanding, there has been a significant amount of simulation work directed toward the modeling of Hall thrusters. Lentz created a one-dimensional numerical model which was able to fairly accurately predict the operating characteristics and plasma parameters for the acceleration channel of a particular Japanese thruster [19]. He began the trend of assuming a Maxwellian distribution of electrons and modeling the various species with a fluidic approximation. Additional one-dimensional analytic work was performed by Noguchi, Martinez-Sanchez, and Ahedo. Their construction is a useful first-stab approximation helpful for analyzing low frequency axial instabilities in SPT-type thrusters [20].

In the two-dimensional realm, Hirakawa was the first to model the Rθ-plane of the thruster channel [15]. Fife's contribution was the creation of an axisymmetric RZ-plane "hybrid PIC" computational model of an SPT thruster acceleration channel [9][10]. This simulation also assumed a Maxwellian distribution of fluidic electrons while explicitly modeling ions and neutrals as particles. The results of this study were encouraging and successfully predicted basic SPT performance parameters. However, they were unable to accurately predict certain details of thruster operation due to their overly strict Maxwellian electron assumption. Roy and Pandey took a finite element approach to the modeling of thruster channel dynamics while attempting to simulate the sputtering of thruster walls [22]. A number of studies have also numerically examined the plasma plume ejected by Hall thrusters and ion engines. At MIT, for example, Murat Celik, Mark Santi, and Shannon Cheng continue to evolve the three-dimensional Hall thruster plume simulation they have already demonstrated [7].

Finally, the most relevant work to the current project was Szabo's development of a two-dimensionl, RZ-plane, Full-PIC SPT model [26] and Blateau and Sullivan's later additions to it [5][25]. This work was able to identify problems with and help redesign the magnetic field of the mini-TAL thruster built by Khayms [16]. In addition, the code was able to predict thrust and specific impulse values for an experimental thruster to within 30%.

## 1.4 Full-PIC Vs. Other Methods

Full-PIC code attempts to model every individual electron, neutral, and ion as a separate entity. It is distinct from a "hybrid PIC" model in that it does not rely on Maxwellian

electron distributions, averaged levels of electron mobility, or wall sheath effects calculated simply from the electron temperature [9][10]. The PIC level of simulation is obviously more accurate and moreover proved necessary when Szabo and Fife attempted to model the Busek BHT-200-X2 and the SPT-100 thrusters with a hybrid-PIC model. They were unable to match experimental levels of Xenon double ionization and hypothesized therefore that the isotropic and Maxwellian electron energy distributions assumed by the hybrid code were incorrect. Thus, full-PIC modeling became desirable [26][27].

## 1.5 Physical Acceleration "Tricks"

Of course, with accuracy comes the price of drastically increased computational requirements. Not only does the PIC model require more particles to be tracked, but physical electron dynamics also happen on a timescale that is a million times shorter than the timescale of the dynamics of the heavier, slower-moving neutrals. Realizing the impossibility of completing useful computations given such a daunting situation, Szabo [26], and others before him [13][14][15], employed several clever numerical techniques to accelerate the physics of the system.

To mediate the first of these issues, Szabo represented all three elementary species, ions, neutrals, and electrons, as superparticles. A superparticle consists of a large number of elementary particles, on the order of $10^6$, that are lumped together into a single computational object. Statistical techniques are then used to ensure that in the limit, superparticles react similarly to the large number of particles they represent. Superparticles were, of course, used in previous hybrid-PIC simulations [14][15], with

the only difference now being that electrons are treated as superparticles along with the heavy particles.

The second issue was dealt with in two different ways; one way sped up the heavy particles and the other slowed the kinetics of the electrons. First, by decreasing the mass of the heavy ion and neutral particles and through careful accounting of this artificial mass factor throughout the remaining calculations, the heavy particles may be sped up with a minimal loss of information. Second, by increasing the free-space permittivity constant, $\varepsilon_0$, the Debye length of the plasma is increased. This allows for better resolution of electron motion on a coarse grid, as well as increasing the timescale of electron dynamics to a level more on par with the heavy particle motion. Szabo took care to keep track of both of these artificial accelerations, and a detailed discussion of their effects on physical processes can be found in [26]. It is enough for our present purposes to believe that these "tricks" have been implemented correctly, and that by using them we are able to approach an equilibrium solution in a tractable, if still not particularly practical, amount of computational time.

## 1.6 Introduction to Parallelization

A parallel computer can be simply defined as any computing machine with more than one central processing unit. The first "multi-processor" computers, such as Bull of France's Gamma 60 can date their development to the late 1950's, originating practically on the heels of single-processor machines [31]. The simple, but powerful idea behind parallelization is perhaps best exemplified by two old adages: "two heads are better than one" and "divide and conquer."

The former adage describes the concept behind the structure of the parallel computer itself. In the search for faster and more powerful computers, it was quickly realized that it might often be more cost-effective to create a large number of cheap, weaker machines and combine them in such a way that they acted like a much more powerful one. The more processors available for use by a programmer, the more computing he should be able to accomplish. Supercomputers like the Cray family of computers and parallel machines like the one used in this research, a 32 processor Compaq Alpha, demonstrate the incredible computational power that can be achieved via parallel construction while the low cost Beowulf clusters that have become popular with universities in recent years exemplify the versatility of this architecture and a cheap, logical extension of it [30].

The second saying is more a description of the programming techniques used by parallel programmers. Many important problems lend themselves extremely well to a "divide and conquer" strategy. Finite element analyses, matrix inversions, Fourier transforms, numerical simulations (including particle-in-cell simulations), and innumerable other problems can be readily broken down into finer pieces which can then be attacked and solved by separate processors. In doing so, we not only decrease the astronomical time required to reach a solution, but in many cases, we are able to reduce the amount of memory required by each processor, another important limiting factor for many applications.

## 1.7 Thesis Objectives

Despite the Herculean efforts at acceleration like those mentioned in Section 1.5, the MIT Full-PIC simulation of a Hall thruster still requires an impractical amount of time to complete, leaving larger, higher power thrusters out of reach. It was our goal to significantly decrease the required computation time through the use of parallelization techniques. In doing so, we hoped to enable more detailed exploration of Hall thrusters and their phenomena by allowing for larger, more realistic simulations and for research involving the optimization of parameters across many different thruster configurations. While accomplishing this, we hoped to demonstrate the viability of these parallel-computing techniques toward the future creation of a three-dimensional Hall thruster PIC code.

## 1.8 Organization of the Thesis

In this thesis, a methodology for parallelizing the MIT PIC simulation will be presented and detailed and the results of the implementation shown. Chapter 2 will discuss some relevant aspects of the serial code which must be understood before their parallel counterparts may be discussed. Chapter 3 begins the parallelization of the code by focusing on the most difficult part, the electric potential calculation. Chapter 4 completes the description of the parallelization for the remainder of the PIC algorithm. The following chapter, Chapter 5, presents some preliminary results and ground-truthing of the parallel implementation. In Chapter 6, two investigations performed using the newly-

parallelized code are described with their attendant results. Finally, Chapter 7 sums up the thesis and presents some future paths further research might explore.

# Chapter 2

# The Serial Code

## 2.1 Summary of Simulation Technique

For complete and detailed explanations of the MIT Hall thruster PIC simulation, Szabo [26], Blateau [5], and Sullivan [25] may be consulted. We will herein provide only a brief overview of the physical problem being simulated and the methodology used to do so. Greater attention will be given to those aspects of the model which bear relevance to the main focus of this work, the parallelization of the simulation.

## 2.2 Structure of the Serial Simulation

A flowchart of the basic serial computation performed by Szabo's code is shown in Figure 2-1. After initial loading of parameters, such as geometric constants, operating voltages, nominal mass flow rates, and the pre-computed, constant-in-time magnetic field, the main iterations begin. The charge distribution is calculated using the positions of present charged particles as well as those charges which have been absorbed by the walls. Next, the electric potential and field are calculated through the use of a Gauss's

Law solver. The solution technique used is described in detail in Section 3.3. With the calculated electric field, the position of present particles can be adjusted using electromagnetic force equations. Concurrently, new particles are injected and numerous types of collisions are incorporated. The final step in the iteration is to calculate overall distributions of moments, such as temperature, and collect other engine performance data, such as thrust and efficiency. Once a specified number of iterations has been completed, the simulation terminates by performing some initial post-processing of data, saving important information to allow possible restart, and clearing allocated memory. Of course, the parallelization effort will focus largely on the portions of the code that are iterated multiple times, the benefits of parallelizing the initialization and post-processing steps being minimal.

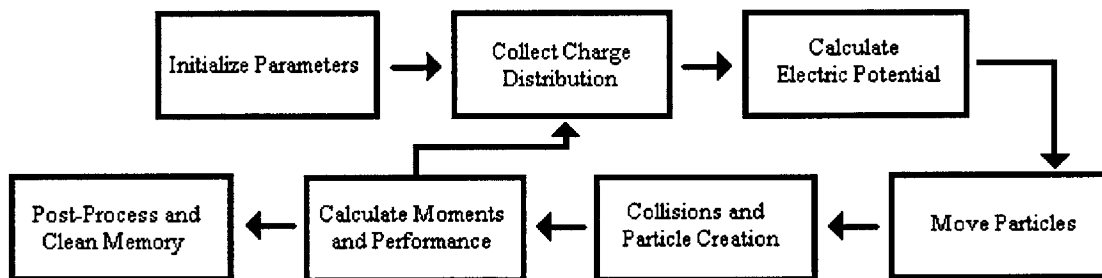



**Figure 2-1: A flowchart of the overall serial code structure.**

## 2.3 Initialization

The serial PIC code requires a significant number of engine parameters, even excluding geometry, to be first loaded from outside data files. In case future researchers wish to reproduce our results, we have included Table 1, listing the particular parameter settings

which were used throughout this work. For a few experiments, particular values were adjusted, such as the number of particles initially simulated. Unless otherwise noted, however, all results shown in this thesis were acquired using the values in Table 1. One minor clarification: the "Temperature of Backstreaming Electrons" referred to in the table is the temperature assigned to those electrons which are created at the right hand simulation boundary in order to maintain the cathode quasi-neutrality condition [26].

**Table 2-1: Parameters used in all trials unless otherwise noted**

| Neutral Mass Flow | 11.41 mg/s | | Anode Potential | 500 V |
|---|---|---|---|---|
| Injected Neutral Temperature | .1 eV | | Cathode Potential | 0 V |
| Mass Factor ($m_n/m_{n(sim)}$) | 1000 | | Boundary Temperature | .06 eV |
| Gamma Factor ($\sqrt{\varepsilon_{0(sim)}/\varepsilon_0}$) | 40 | | Temperature of Backstreaming Electrons | .2 eV |
| Ceramic Dielectric Constant | 4.4 | | Temperature of Free Space Electrons | 2.5 eV |
| Initial Number of Electrons and Ions | 20,000 | | Temperature of Anode Surface | .1 eV |
| Maximum B-Field | 290.664 G | | Save Frequency | 500 iterations |

In addition to the loading of various constants, the magnetic field structure must also be loaded and interpolated to our computational grid. The nominal magnetic field used for the P5 thruster modeled is shown in Figure 2-2. In her thesis, Sullivan indicated that this magnetic field is prone to severe magnetic mirroring effects which tend to force the chamber plasma away from the axially-inner wall of the channel [25]. This is important to recognize when analyzing the results of our experiments, and future work should perhaps investigate the effects of an altered magnetic field.

This section of the code was not targeted for parallelization since it is only executed once per simulation. Of course, many portions of it had to be adjusted or

completely rewritten in order to accommodate the parallelization. The initial loading or distributing of particles across processors, for instance, required a parallel implementation which will be discussed in section 4.1.

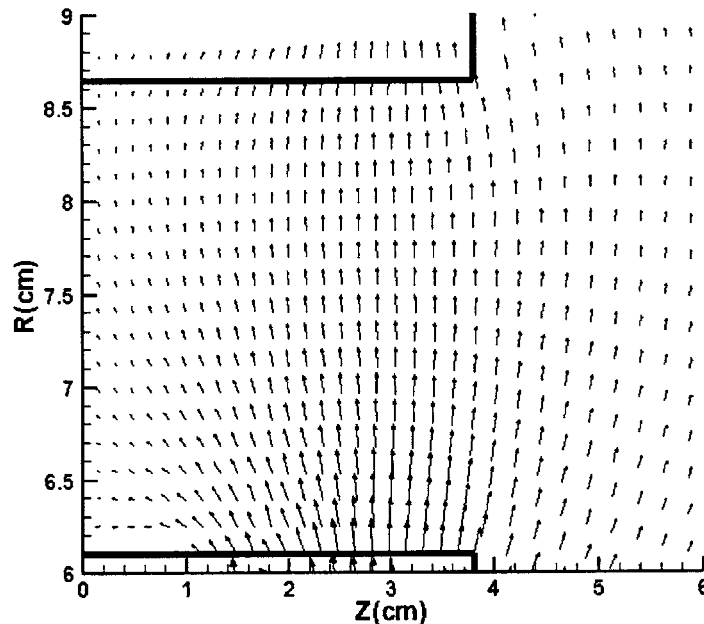



**Figure 2-2: Magnetic field in the chamber of the P5 thruster.**

## 2.4 Electric Potential Calculation

Given a charge distribution, the potential associated with it can be calculated by applying Gauss's Law. This calculation requires a finite volume approximation to derivatives in the region surrounding a grid point. As such, a linearized matrix equation, $A\Phi = Q$, arises. In our case, this equation was solved using the Successive Over-Relaxation technique. The methodology behind both the serial and parallel Gauss's Law solvers will be discussed in more detail in Section 3.3.

## 2.5 Particle Movement

Individual charged particles are subject to various forces in a Hall thruster including electrical, magnetic, and collisional. Collisonal forces will be discussed in the next section. Here we outline the motion of a charged particle given the magnetic and electric fields of a Hall thruster.

### 2.5.1 The Gyro Radius

The well-known Lorentzian force on a charged particle is given by:

$$\vec{F} = q\left(\vec{E} + \vec{v} \times \vec{B}\right) \tag{2.1}$$

In a constant magnitude and direction magnetic field, the $\vec{v} \times \vec{B}$ force is perpendicular to lines of magnetic field and causes a particle to gyrate at the constant frequency:

$$\omega = \frac{qB}{m} \tag{2.2}$$

In this situation, the centrifugal force must balance the magnetic force in order to maintain a constant radius orbit. This balance applied to electrons gives rise to the equation:

$$\frac{m_e v_e^2}{R_e} = e\left(\vec{v}_e \times \vec{B}\right) \tag{2.3}$$

which implies that the gyration, or Larmour, radius for electrons is:

$$R_e = \frac{m_e v_e}{eB} = \frac{v_e}{\omega_e} \tag{2.4}$$

A similar derivation for ions finds that:

$$R_i = \frac{m_i v_i}{eB} = \frac{v_i}{\omega_i} \tag{2.5}$$

From these two equations, one can easily see that the ratio of ion radius to electron radius is proportional to $m_i v_i / m_e v_e$ . If we take the average ion and electron velocities to be the usual expressions for average particle velocity in a Maxwellian distribution:

$$\langle v \rangle = \sqrt{\frac{8k_b T}{m\pi}} \tag{2.6}$$

we find that the ratio of the ion gyro radius to the electron gyro radius is approximately:

$$\frac{R_i}{R_e} \propto \sqrt{\frac{m_i}{m_e}} \tag{2.7}$$

This ratio is of course very large, and so on the dimensions of a Hall thruster, the effect of the magnetic field on ion motion is practically negligible while the electron motion is dominated by the $\vec{v} \times \vec{B}$ forces. Thus, the electrons will mostly be axially trapped inside the acceleration channel of the thruster while the ions are accelerated outward by the applied electric field.

### 2.5.2 Calculation of New Particle Velocity

The simulation is 2.5-dimensional. This means that velocities in the theta direction are retained, but displacements resulting from them are folded back into the r-z plane. The actual calculation of particle motion is performed using a leapfrog algorithm [26]. A particle is first subjected to half a time-step acceleration by the electric field, then a full time-step of rotation in the magnetic field, and finally the second half time-step acceleration in the electric field. There are, of course, errors associated with this method,

but the simulation time-step is kept sufficiently short (on the order of one third of the electron gyro-period) to ensure that these errors are negligible.

## 2.6 Modeled Collisions

Szabo [26] performed a detailed mean free path analysis which enabled him to choose the relevant particle collision possibilities for the simulation. This analysis determined that single and double ionization of neutrals, double ionization of singly-charged ions, ion-neutral scattering, and anomalous Bohm diffusion-causing electron collisions were necessary. The collisions are modeled using a Monte Carlo procedure. The probability of a particle undergoing a certain type of collision can be given by a Poisson distribution:

$$P(collision) = 1 - \exp\left[(n_{slow} v_{fast} \sigma)\Delta t\right] \qquad (2.8)$$

where $n_{slow}$ is the density of the slower-moving particles interpolated to the collision location, $v_{fast}$ is the velocity of the faster moving particle, and $\sigma$ represents the energy-dependent cross-section for the particular type of collision. In normal operation of the code, the term inside the exponential is kept very small to exclude multiple collisions per time step.

## 2.7 Data Collection

After each iteration of particle motion and field calculation has occurred, the simulation calculates critical information about the thruster's operation. The temperatures and densities of each species are saved, as are the calculated electric field and potential. In terms of engine performance, the specific impulse, thrust, actual mass flow rate, and the

various types of relevant currents are also calculated and stored. Finally, the distributions of particles, both those free and those retained by the walls, are stored in order to ensure the simulation may be restarted. A more detailed description of what information is saved by the simulation and how it is calculated can be found in Szabo's thesis [26].

# Chapter 3

# Parallelization of the Code

## 3.1 Resources used

Unless otherwise noted, the results reported in this thesis are from experiments conducted using DeltaSearch Labs 32 processor Compaq Alpha machine. This machine operates under the Tru64 Unix environment. Care was taken to ensure that when timing results was an issue, no other threads were running on the utilized processors.

Data processing and plotting was performed using both Matlab™ and Tecplot™. Microsoft's Visual C++ was used for programming purposes.

## 3.2 Introduction to MPI and MPICH

The original problem we were asked to solve was to increase the computational speed of Szabo's plasma simulation [26] by parallelization of the code for an inhomogeneous cluster of typical desktop Pentiums (i.e. the all-purpose computers in our lab's network). As such, it was necessary to use a highly portable, preferably freely-available

communications protocol that did not rely on shared memory for inter-process communication. The Message Passing Interface standard (MPI) exhibited exactly the combination of attributes we sought.

MPI refers to a library specification developed in the late 1990's by a committee representing parallel software users, writers, and vendors. It dictates standards that must be met by software claiming to implement MPI [32]. One of the most common implementations of this standard, and one which is freely distributed, is known as MPICH. The libraries implemented in MPICH provide high-level tools for the transmission of data between processors that may be networked in numerous different ways, including in our case Ethernet connections and massively parallel machines like the Alpha. These high-level tools allow the programmer to focus on the problem at hand while leaving the mundane details of efficient communication routines to MPI. Because this type of parallel computation requires messages to be sent between processes, it is, of course, not expected to be as efficient or fast as the shared memory model, but the possibility of conducting simulations cheaply on our own network helped to balance the foreseen cost in speed.

Being that this is an Aerospace thesis, the reader may not be very familiar with MPI or message-passing among networks in general, so it may be instructive to explain a few of the basic MPICH functions used in this project and their general usage information. Appendix A at the end of this work does just that and is intended to be a handy reference for anyone who may wish to build upon this work in the future. Complete documentation of MPICH can also be found at [32].

## 3.3 Gauss's Law Solver

In order to estimate the electric field at each point in the simulation region, we do not solve Poisson's equation as would be the normal approach. Instead, we start from the integral form of Gauss's Law which is:

$$\oint \vec{E} \cdot d\vec{s} = \frac{1}{\varepsilon_0} Q \qquad (3.1)$$

In CGS units this is the familiar:

$$\oint \vec{E} \cdot d\vec{s} = 4\pi Q \qquad (3.2)$$

The left hand side integral denotes the flux of electric field across a boundary, in our case the boundary of a single simulation cell, and the right hand side is the scaled charge contained inside that cell. The electric field is related in our cylindrical coordinates to the potential by:

$$\vec{E} = -\nabla\Phi = -\frac{\partial\phi}{\partial z}\hat{e}_z - \frac{\partial\phi}{\partial r}\hat{e}_r - \frac{1}{r}\frac{\partial\phi}{\partial\theta}\hat{e}_\theta \qquad (3.3)$$

In our case $d/d\theta = 0$ because of our axisymmetric assumption. The derivatives are then calculated using the chain rule:

$$\frac{\partial\phi}{\partial r} = \frac{\partial\phi}{\partial\xi}\frac{\partial\xi}{\partial r} + \frac{\partial\phi}{\partial\eta}\frac{\partial\eta}{\partial r} \qquad (3.4)$$

$$\frac{\partial\phi}{\partial z} = \frac{\partial\phi}{\partial\xi}\frac{\partial\xi}{\partial z} + \frac{\partial\phi}{\partial\eta}\frac{\partial\eta}{\partial z} \qquad (3.5)$$

where $\xi,\eta$ represent the transformed computational coordinates. Since $\xi_z$, $\xi_r$, $\eta_z$, and $\eta_r$ depend only upon the geometry of our grid, they are pre-computed just once at the beginning of the simulation to save time. The well-known formula for these calculations is given by:

$$\begin{bmatrix} \xi_z & \xi_r \\ \eta_z & \eta_r \end{bmatrix} = \begin{bmatrix} z_\xi & z_\eta \\ r_\xi & r_\eta \end{bmatrix}^{-1} = \frac{1}{z_\xi r_\eta - z_\eta r_\xi} \begin{bmatrix} r_\eta & -z_\eta \\ -r_\xi & z_\xi \end{bmatrix} \tag{3.6}$$

where, for example, $z_\xi$ for grid cell (i,j) can be approximated by a standard differencing:

$$z_\xi(i,j) = \frac{1}{2}\left(z_{i+1,j} - z_{i-1,j}\right) \tag{3.7}$$

Then if we wish to calculate the electric field flux through, for instance, the $+\eta$ face of a computational cell centered on node (i, j), we can approximate the $\eta$ derivatives by a simple first-order differencing scheme:

$$\left(\frac{\partial\phi}{\partial\eta}\frac{\partial\eta}{\partial r}\right)_{i,j+\frac{1}{2}} = \left(\phi_{i,j+1} - \phi_{i,j}\right)\overline{\eta}_r \tag{3.8}$$

$$\left(\frac{\partial\phi}{\partial\eta}\frac{\partial\eta}{\partial z}\right)_{i,j+\frac{1}{2}} = \left(\phi_{i,j+1} - \phi_{i,j}\right)\overline{\eta}_z \tag{3.9}$$

$$\overline{\eta}_z = \frac{1}{2}\left(\eta_z\right)_{i,j} + \frac{1}{2}\left(\eta_z\right)_{i,j+1} \tag{3.10}$$

$$\overline{\eta}_r = \frac{1}{2}\left(\eta_r\right)_{i,j} + \frac{1}{2}\left(\eta_r\right)_{i,j+1} \tag{3.11}$$

By averaging their values at the corners, the $\xi$ derivatives can also be obtained by:

$$\left(\frac{\partial\phi}{\partial\xi}\frac{\partial\xi}{\partial r}\right)_{i,j+\frac{1}{2}} = \frac{1}{2}\frac{\phi_{i+1,j+1} - \phi_{i-1,j+1}}{2}\left(\frac{d\xi}{dr}\right)_{i,j+1} + \frac{1}{2}\frac{\phi_{i+1,j} - \phi_{i-1,j}}{2}\left(\frac{d\xi}{dr}\right)_{i,j} \tag{3.12}$$

$$\left(\frac{\partial\phi}{\partial\xi}\frac{\partial\xi}{\partial z}\right)_{i,j+\frac{1}{2}} = \frac{1}{2}\frac{\phi_{i+1,j+1} - \phi_{i-1,j+1}}{2}\left(\frac{d\xi}{dz}\right)_{i,j+1} + \frac{1}{2}\frac{\phi_{i+1,j} - \phi_{i-1,j}}{2}\left(\frac{d\xi}{dz}\right)_{i,j} \tag{3.13}$$

With similar approximations, we can compile the fluxes on all four sides into a single equation for $\phi_{ij}$ dependent only on itself, the values of $\phi$ in the eight surrounding nodes, and the unchanging geometric constants of those nodes. This then allows us to estimate a new potential via:

$$\phi_{i,j}^{k+.5} = \frac{C}{N+S+E+W} \tag{3.14}$$

$$
\begin{aligned}
C = {} & N\phi_{i,j+1} + S\phi_{k,j-1} + E\phi_{k+1,j} + W\phi_{k-1,j} + q_{i,j} + \\
& U\frac{1}{4}\left(\phi_{i+1,j} + \phi_{i+1,j+1} - \phi_{i-1,j} - \phi_{i-1,j+1}\right) + \\
& D\frac{1}{4}\left(\phi_{i+1,j} + \phi_{i+1,j-1} - \phi_{i-1,j} - \phi_{i-1,j-1}\right) + \\
& R\frac{1}{4}\left(\phi_{i+1,j+1} + \phi_{i,j+1} - \phi_{i+1,j-1} - \phi_{i,j-1}\right) + \\
& L\frac{1}{4}\left(\phi_{i-1,j+1} + \phi_{i,j+1} - \phi_{i-1,j-1} - \phi_{i,j-1}\right)
\end{aligned}
\tag{3.15}
$$

$$
\phi_{ij}^{k+1} = \phi_{ij}^{k} + \omega\left(\phi_{ij}^{k+.5} - \phi_{ij}^{k}\right) \tag{3.16}
$$

where $\omega$ is defined as the over-relaxation factor, may range between 0 and 2, and will be discussed more in later sections. The constants that depend only upon geometry are pre-computed and can be given as:

$$
\begin{aligned}
E &= \left|\left(\xi_z\vec{i}_z + \xi_r\vec{i}_r\right)_E \cdot \vec{n}_E A_E\right| \\
W &= \left|\left(\xi_z\vec{i}_z + \xi_r\vec{i}_r\right)_W \cdot \vec{n}_W A_W\right| \\
N &= \left|\left(\eta_z\vec{i}_z + \eta_r\vec{i}_r\right)_N \cdot \vec{n}_N A_N\right| \\
S &= \left|\left(\eta_z\vec{i}_z + \eta_r\vec{i}_r\right)_S \cdot \vec{n}_S A_S\right|
\end{aligned}
\tag{3.17}
$$

$$
\begin{aligned}
R &= \frac{1}{2}\left[\left(\frac{\partial\eta}{\partial z}_k + \frac{\partial\eta}{\partial z}_{k+1}\right)\vec{i}_z + \left(\frac{\partial\eta}{\partial r}_k + \frac{\partial\eta}{\partial r}_{k+1}\right)\vec{i}_r\right] \cdot \vec{n}_E A_E \\
L &= \frac{1}{2}\left[\left(\frac{\partial\eta}{\partial z}_k + \frac{\partial\eta}{\partial z}_{k-1}\right)\vec{i}_z + \left(\frac{\partial\eta}{\partial r}_k + \frac{\partial\eta}{\partial r}_{k-1}\right)\vec{i}_r\right] \cdot \vec{n}_W A_W \\
U &= \frac{1}{2}\left[\left(\frac{\partial\xi}{\partial z}_j + \frac{\partial\xi}{\partial z}_{j+1}\right)\vec{i}_z + \left(\frac{\partial\xi}{\partial r}_j + \frac{\partial\xi}{\partial r}_{j+1}\right)\vec{i}_r\right] \cdot \vec{n}_N A_N \\
D &= \frac{1}{2}\left[\left(\frac{\partial\xi}{\partial z}_j + \frac{\partial\xi}{\partial z}_{j-1}\right)\vec{i}_z + \left(\frac{\partial\xi}{\partial r}_j + \frac{\partial\xi}{\partial r}_{j-1}\right)\vec{i}_r\right] \cdot \vec{n}_S A_S
\end{aligned}
\tag{3.18}
$$

The over-relaxation is said to have converged when the residue:

$$RHS = \frac{\left(\phi_{ij}^{k+1} - \phi_{ij}^{k+.5}\right)}{(N+S+E+W)^{-1}} = \frac{(\omega-1)\left(\phi_{ij}^{k+.5} - \phi_{ij}^{k}\right)}{(N+S+E+W)^{-1}} \quad (3.19)$$

becomes less than some small pre-defined value $\varepsilon$ at every point throughout the entire simulation domain.

## 3.4 Parallelization of Successive Over-Relaxation

When the above equations for $\phi_{ij}$ are compiled at each node into a single system, the simple equation below is obtained:

$$A\Phi = Q \quad (3.20)$$

Where in our case, the matrix A corresponds to the coefficients of the linear derivative approximations along with the geometric constants, $\Phi$ is a N_R*N_Z by 1 vector of the $\phi$ values at the node points, and Q is the N_R*N_Z by 1 vector of the charge values at the node points.

Successive Over-Relaxation (SOR) is the technique that was chosen to iteratively solve equation *(3.20)* for our simulation. The SOR method approximates a new $\Phi^{(k+1)}$ from the previous $\Phi^{k}$ by equation *(3.16)* which in matrix form is given by:

$$\Phi^{(k+1)} = \omega D^{-1}(Q - L\Phi^{(k+1)} - R\Phi^{k}) + (1-\omega)\Phi^{k} \quad (3.21)$$

where D, L, and R, are, respectively, the diagonal, left-, and right-triangular matrices of A, $\omega$ is the over-relaxation factor, and Q is again the vector of charges [8]. For SOR to converge, $\omega$ must be chosen between 0 and 2, with $\omega=1$ reducing SOR to the simpler Gauss-Siedel iteration. At first glance, it would appear that this method is inherently serial. As shown in Figure 3-1, because of the $L\Phi^{(k+1)}$ term, in order to calculate a new

solution vector at point (i.j), we would need to have already calculated new solution vectors at each point (m,n) where m ≤ i and n < m.

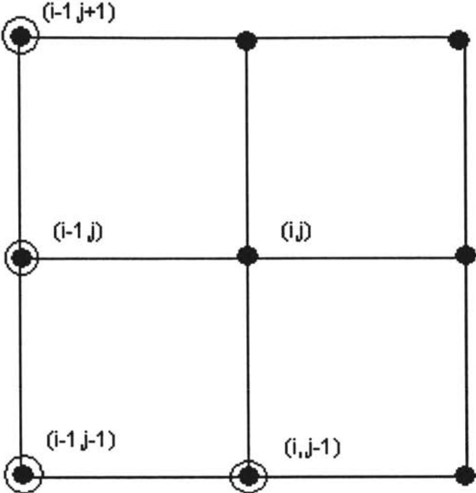



**Figure 3-1: The Serial Nature of SOR. To calculate $\Phi^{(k+1)}$ at point (i,j), we need to have already calculated $\Phi^{(k+1)}$ below and to the left of (i,j).**

Fortunately, this problem is well-known and is often solved by a solution method known as the red-black SOR algorithm which groups the nodes in a clever way so that the matrix equations above become decoupled and the algorithm becomes much easier to parallelize [29]. In a typical red-black scheme, half the points are "colored" black and the other half red in a checkerboard-like fashion. (Figure 3-2). Then a typical iteration would update all of the black points first using the previous red values. Next the red points would be updated using the new values calculated at the black points. This would constitute one complete iteration and yield a $\Phi^{(k+1)}$ not unlike that achieved by equation *(3.21)* above. Of course, this scheme does not exactly preserve equation *(3.21)*, as the black points will effectively always be an iteration behind the red points. However, Kuo and Chan have shown that while this does slightly adversely affect the convergence rate

of the algorithm, the difference is small when compared to the benefits obtained through parallelization [18]. It should be noted, though, that this is one reason that we cannot expect to achieve linear speed up; the more processors we use, the more nodes we will be calculating with the slightly asynchronous boundary data, and the worse the convergence rate could become.

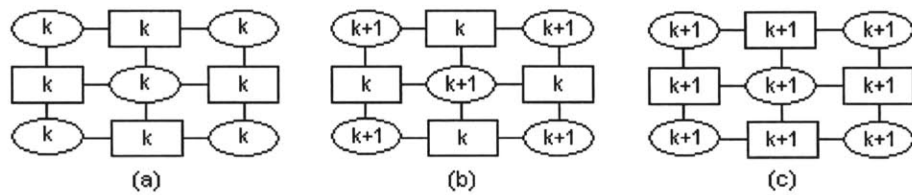



**Figure 3-2: Red-Black SOR Algorithm with ovals representing black nodes and rectangles representing red. (a) The $k^{th}$ iteration begins. (b) The black nodes update first using the red values from the $k^{th}$ iteration. (c) The red nodes update using the black values from the $(k+1)^{th}$ iteration.**

We adapted the red/black scheme slightly into something more reminiscent of a zebra or strip SOR scheme that in many cases, including ours, requires less passing of data between processors [2]. Each processor is first assigned a strip of the solution region which will be its responsibility to solve. (Figure 3-3(a)). If *np* is the number of processors being utilized and *pid* is a unique integer between 0 and *np*-1 identifying each processor, we defined processor *pid*'s solution region to be those points (i, j) such that:

$$pid < NR \bmod np \Rightarrow \begin{Bmatrix} j \geq \left\lfloor \dfrac{NR}{np} \right\rfloor * pid + pid \\ j \leq \left\lfloor \dfrac{NR}{np} \right\rfloor * (pid+1) + pid \end{Bmatrix}$$

$$pid \geq NR \bmod np \Rightarrow \begin{Bmatrix} j \geq \left\lfloor \dfrac{NR}{np} \right\rfloor * pid + NR \bmod np \\ j \leq \left\lfloor \dfrac{NR}{np} \right\rfloor * (pid+1) + NR \bmod np \end{Bmatrix} \tag{3.22}$$

$$0 \leq i \leq NZ - 1$$

where NR is the number of grid cells in the azimuthal direction and NZ is the number of grid cells in the radial direction. The "black" points in this algorithm are then taken to be the points in the top of each processor's solution strip. These points are the first to be updated during an iteration, and we update them serially from left to right. The values from the $k^{th}$ iteration are used in the calculation for $\Phi_{ij}^{(k+1)}$ except for the value at (i-1,j) for which is used the freshly calculated value. Once calculated, the updated black values are sent to the processor's neighbor to the north. Now the "red" values are updated. This includes all of the remaining points in the solution region. These are calculated from top to bottom first and then across the strip. (Figure 3-3(c)). In this way, a $(k+1)^{th}$ solution is obtained.

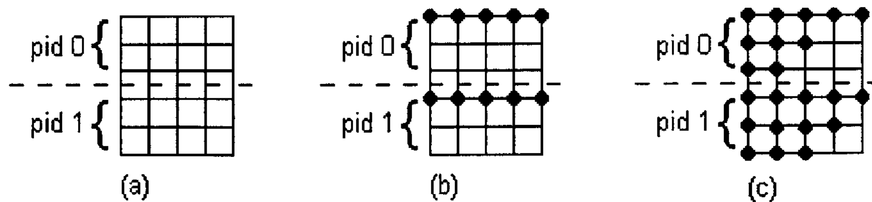



**Figure 3-3: Parallelizing SOR. (a) Each processor receives a strip of nodes. (b) The top row of nodes is calculated first. These are the "black" nodes. (c) The remaining "red" nodes are then calculated from top to bottom first, then from left to right.**

## 3.5 Implementation of Gauss's Law Parallelization Using MPICH

Given the "lopsided zebra" algorithm described above, it was next necessary to translate these ideas into MPICH language. In particular, we needed to decide the quantity of data to store on each processor and how and when to transfer data between processors efficiently.

The basic architecture of the program was one of a single master processor overseeing some number of slave processors. In this case, the master process performs all of the necessary initializations, loading of previous data, and calculation of geometric grid constants. It then broadcasts this information to each of the slaves, signaling them to begin synchronously calculating the electric potential in their respective regions. When the slaves have finished, they must transfer their results back to the master process which then continues on to the remainder of the code. The slaves must then loop back and wait for the master to signal them on the next iteration.

Given this structure, we needed to understand exactly what information, from the master and from its counterparts, a slave process would require to calculate the electric potential and what, other than the calculated potential, the slaves should transmit back to the master process. For instance, equation *(3.22)* shows that each processor must store the charge distribution at the current time step for at least the cells in its region. This value does not change during the computation of the electric potential, but does change every time step and so must be rebroadcast by the master to each slave at every iteration. A less costly requirement is that every slave must store the geometric values, N, S, E, W, and U, D, L, R for the nodes within its solution region plus and minus one node. Since

our mesh is not adaptive, these do not change throughout the simulation, and so must be sent only once to each of the slaves.

Moving deeper into the zebra algorithm, equation *(3.22)* again shows us that the "black" nodes require the $\phi$ values of the previous over-relaxation iteration at the nodes one row above them and one row below. This means that at every iteration a typical processor, excluding processor 0 whose assigned region is the bottom of the simulation region, must send its bottom row of values to its neighbor to the south. Once these black nodes have finished calculating their values, the red nodes will begin to calculate the rest of the assigned space. However, they will require the $\phi$ values at the black nodes of the processor neighboring them to the south. For this reason, at each over-relaxation iteration, a typical processor, excluding the topmost, must send its top row of $\phi$ values to the processor neighboring it to the north. In our case, this task is made slightly more difficult by the fact that we have a gradient boundary condition on the right hand side of the simulation region. In order for proper convergence, therefore, we must calculate all right hand side nodes with the most current possible values. As such, once the red values have been calculated at all nodes except the right hand side nodes, we must calculate the right hand side black value using the most current red node values. Finally, when the over-relaxation has converged, it can be seen that a slave will only know the electric potential for the current time step at points in its assigned region and perhaps one node above and below. In fact, a typical slave, that is a slave whose calculation region neighbors exactly two other slaves' regions, will, at the end of the potential calculation, have the proper values of $\phi$ for its entire solution region and for one row beneath. It will not have the proper values for the row above its region or any of the remaining portions

of the grid. Therefore, the potential in at least the row above the solution region must be sent to the processor by its northerly neighbor so the proper values will be held when the next time step begins. Admittedly, the potential values by this point are vanishingly similar between time steps and this final communication may appear wasteful, but one could easily imagine that with large numbers of processors and over long simulation durations, even this tiny error could grow to be significant.

In addition to these major communications, there are a few more minor ones to be dealt with as well. Convergence, and thus the stopping condition of the solver, is satisfied when the maximum value of RHS, given in equation *(3.19)*, over all processors falls below $\varepsilon$. Therefore, at each iteration, we must reduce the value of RHS over the slave processors and find its overall maximum. If this maximum is less than $\varepsilon$, the master process, also known as the root, must broadcast a flag back to the remaining slaves, signaling them to cease the computation for this time step. Finally, there are various other control values and constants, such as the total number of simulation iterations desired and the choice of the relaxation parameter, $\omega$, to use which must be communicated at differing intervals throughout the simulation. The table below helps to summarize the ideas presented in this previous section.

**Table 3-1: Summary of the major communications for the Gauss's Law solver.**

| Variable | Sent From | Received By | Transmission Frequency | MPI Routine(s) Used |
|---|---|---|---|---|
| N,S,E,W,U,D,L,R | Master | All Slaves | Simulation | Bcast |
| Charge | Master | All Slaves | Time Step | Bcast |
| Black Nodes' $\phi$ | *pid*-1 | *pid* | Relaxation Iter. | Send/Recv/ Sendrecv |
| Bottom Row of Red Nodes' $\phi$ | *pid*+1 | *pid* | Relaxation Iter. | Send/Recv/ Sendrecv |
| RHS | All Slaves | Master | Relaxation Iter. | Reduce (Sum) |
| Completion Flag | Master | All Slaves | Relaxation Iter. | Bcast |

From this analysis, it can quickly be seen that the most time-consuming communication involved in the process will be the transmission of boundary node information at each iteration during the process. Luckily, MPICH makes available the special function Sendrecv for exactly such a task as ours which helps to speed up the process by only requiring in most cases one function call per transfer and by allowing the sends and receives to proceed in either order. Use of this function provides a not insignificant time savings at each relaxation iteration which translates into a much larger savings on the whole.

## 3.6 Preliminary Results For Gauss's Law Parallelization

It was believed that the iterative solution of the Gauss's Law matrix equation was the greatest single time constraint for the simulation. Therefore, it was the first portion to be parallelized.

Some results of the simulation operating in serial mode except for the Gauss's Law solution portion are shown in Figures 3-4 and 3-5 compared to results of a completely serial simulation. Both simulations were allowed to run for 140,000

iterations, operated at 300 volts, and were seeded with 20,000 each of electrons, neutrals, and ions. The time-averaged electric field calculated in both cases was very similar, as were indicative parameters like the electron temperature. The ionization region is clearly visible in the electron temperature plots for both trials. While the plots do show great similarity, we were bothered that any dissimilarities at all were present. Eventually we discovered that both the serial and the parallel codes were not converging for some iterations during the trials and that this was the cause of these observed differences. More on the results of our investigations into these dissimilarities can be found in Section 5.6.

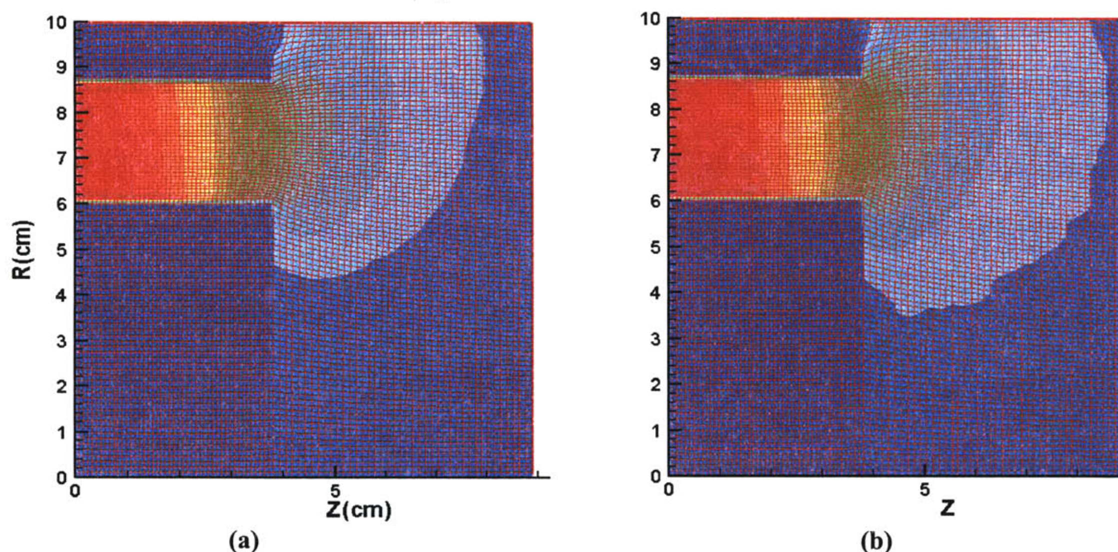



**Figure 3-4: Average Electric Potential of (a) Serial Code and (b) Parallel Code Over Similar Experiments.**

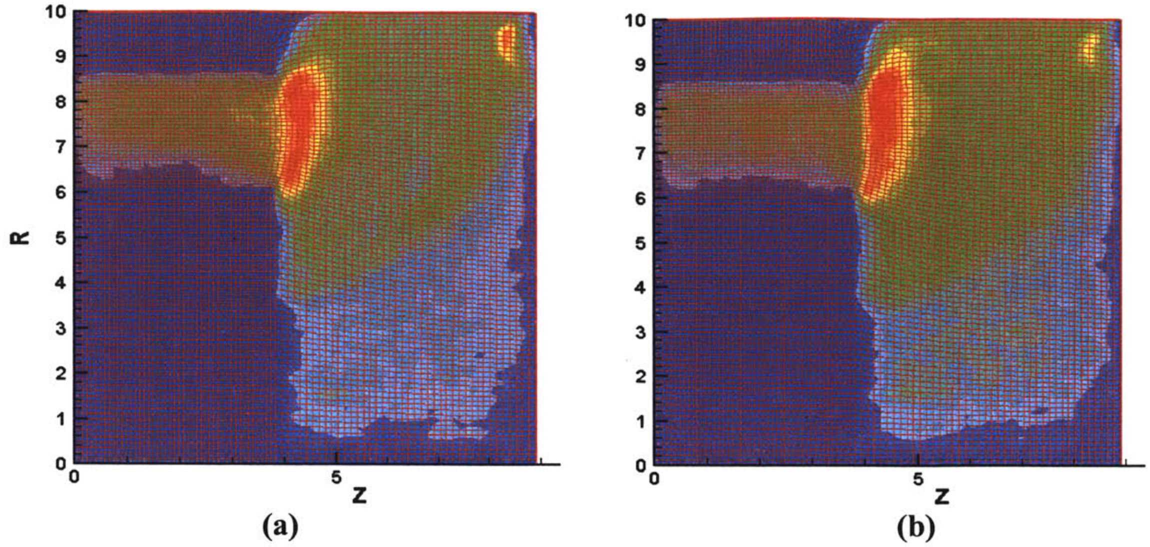



**Figure 3-5: Average Electron Temperatures of (a) the serial code and (b) the parallel code.**

## 3.7 Ground-Truthing of the Gauss's Law Solver

During the preliminary trials, slight differences were noticed between the potentials and temperatures calculated by the parallel solver and the serial solver. As mentioned above, the reasons for these differences were only later discovered and will be discussed in section 5.6. It was important to us, however, to test the accuracy of the parallel solver on a known problem with a known solution to determine if the solver was indeed working correctly.

Following [26], we therefore defined an analytical target potential P by the smooth, continuous, periodic function:

$$P = A_z \cos\left(\frac{2\pi z}{T_z}\right) + A_r \cos\left(\frac{2\pi r}{T_r}\right) \quad (3.23)$$

$$\vec{\nabla} P = -A_z \sin\left(\frac{2\pi z}{T_z}\right)\left(\frac{2\pi}{T_z}\right)\vec{i}_z - A_r \sin\left(\frac{2\pi r}{T_r}\right)\left(\frac{2\pi}{T_r}\right)\vec{i}_r \quad (3.24)$$

$$\nabla^2 P = -A_z \cos\left(\frac{2\pi z}{T_z}\right)\left(\frac{2\pi}{T_z}\right)^2 - A_r r \cos\left(\frac{2\pi r}{T_r}\right)\left(\frac{2\pi}{T_r}\right)^2 - A_r \sin\left(\frac{2\pi}{T_r}\right)\left(\frac{2\pi}{T_r}\right) \quad (3.25)$$

Given this potential, the charge distribution that gave rise to it was then analytically calculated simply by setting $\varepsilon = 1$ and using Poisson's equation with:

$$\rho(z,r) = -\nabla^2 P(z,r) \tag{3.26}$$

We next approximated this charge distribution and discretized it in a form usable by the Gauss's Law function by taking its value at each grid node and multiplying it by the volume that surrounds that node. Plots of this potential and charge are shown in Figure 3-6. This distribution was then passed to the Gauss's Law function which used it and the value of P at the boundaries of the solution region to calculate an approximation of the potential, $\Phi$. By comparing the differences between $\Phi$ and P we were able to deduce a measure of the accuracy of the potential solver given various levels of parallelization.

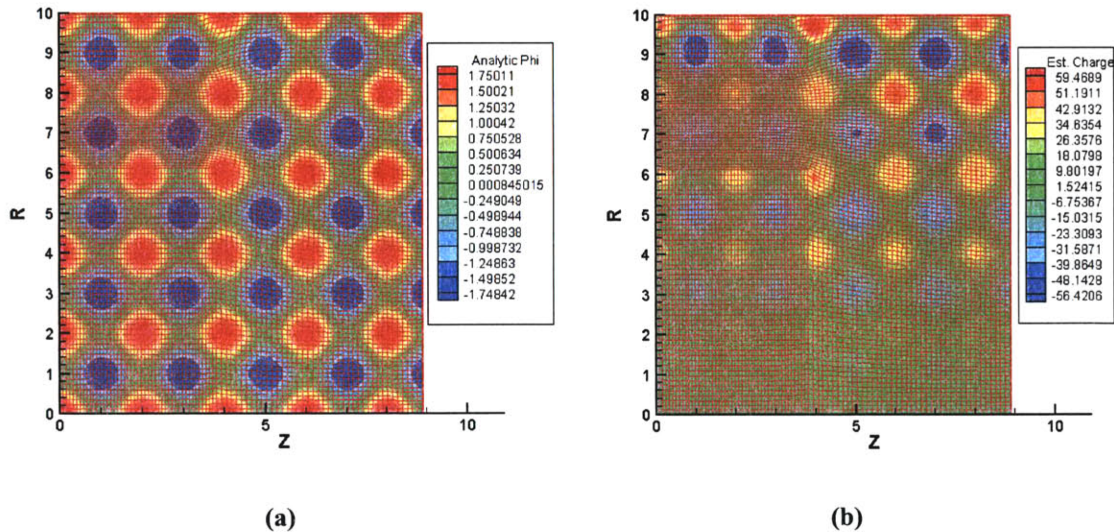



**Figure 3-6: The Analytic (a) potential and (b) charge used in the Ground Truth tests.**

The approximate potential $\Phi$ was calculated in this way for various numbers of processors. To visualize these results, a normalized difference

$$\Delta = \frac{\left|\phi_{i,j} - P_{i,j}\right|}{\max_{i,j}(\phi)} \tag{3.27}$$

was calculated for each point and plotted in Figure 3-7. It was observed that the parallel algorithm and the serial algorithm performed almost exactly the same. In fact, the potential that was calculated using 32 processors differed from that calculated serially only in the tenth decimal place; this was the maximum deviation from the serial calculation and was within the range expected for 8-byte precision numbers. Overall the accuracy of the potential solver was seen to be quite good, with the maximum $\Delta$ of about .065 occurring only at one particular, highly non-uniform cell in the grid. Aside from this small region, all other errors were accurate to at worst 1 or 1.5%. Given a much simpler mesh, Szabo detected only slightly lesser levels of inaccuracy [26].

The similar level of accuracy between the serial and parallel potential solvers on these simple charge distributions seemed to contradict the larger differences observed in the time-averaged electric potentials shown above in Figure 3-4. This issue was indeed worrisome, and provided impetus for the investigation described in Section 5.6 which uncovered an important stability problem in the potential solver. This stability problem rather than an unequal level of solver accuracy proved to be the major factor in the differences seen between the time-averaged serial and parallel potentials.

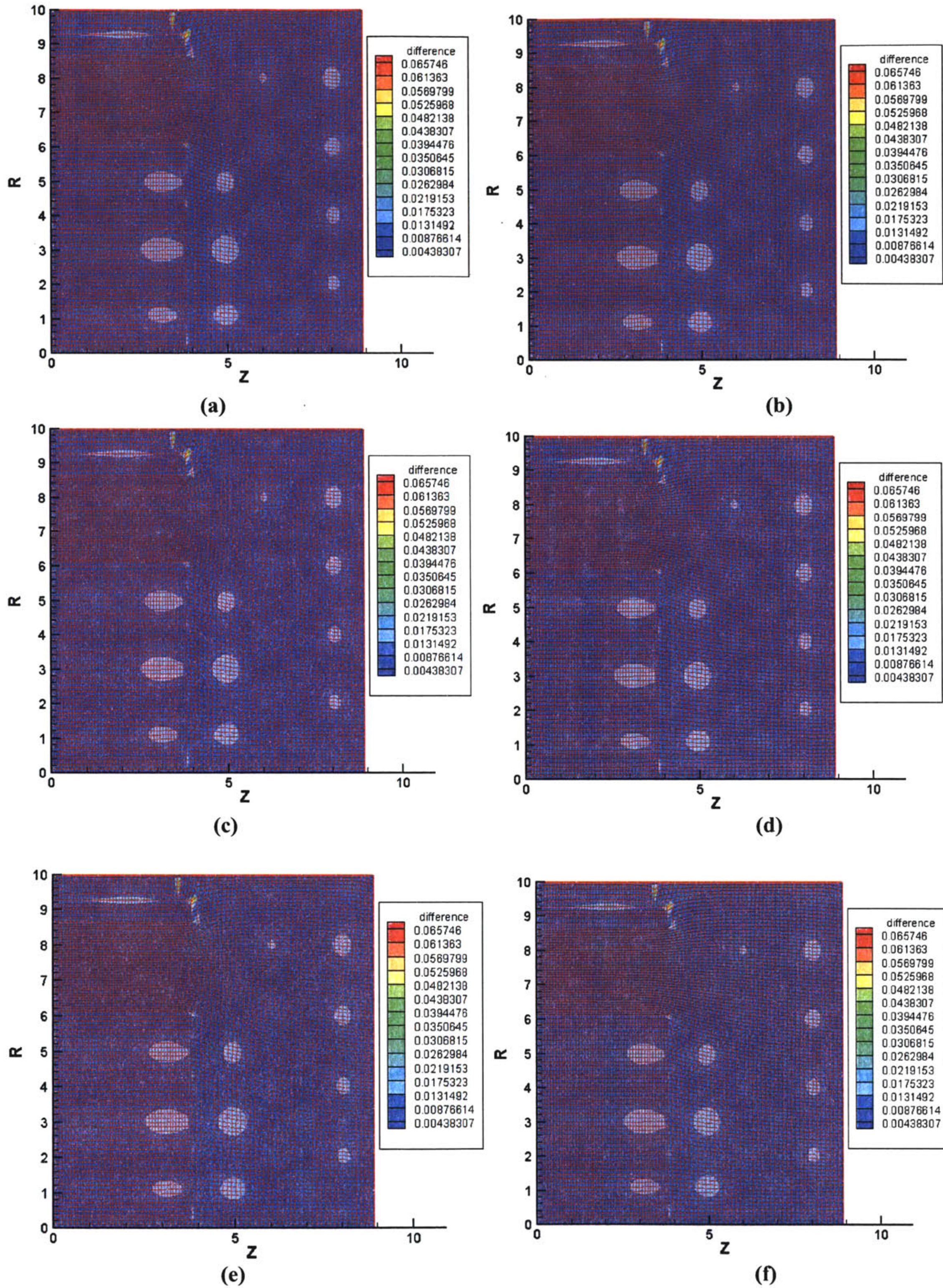



**Figure 3-7: The normalized magnitude of the difference between the analytic and calculated potentials was found for (a) 1 (b) 2 (c) 4 (d) 8 (e) 16 (f) 32 processors. The calculated potentials agreed to one another to the ninth decimal place.**

## 3.8 Further Ground-Truthing of Gauss's Law Solver

While instructive as to the overall accuracy of the nine-point Gauss's Law solver that had been concocted in [26], the above results did not clearly display the alterations in the calculated potential which arose from the parallelization of the code. The errors of parallelization were simply too small and were swamped by the overall computational errors. Therefore, a slightly different test was suggested which would nullify the errors incurred by the nine-point approximation scheme, leaving only errors of precision, convergence, and, of course, parallelization.

The same analytic potential as in equation *(3.23)* was again explored. This time, though, we did not analytically calculate the second derivative of the potential and take its values at the nodes to be the charge input to the solver. Instead, the charge was estimated from the analytic potential using the same nine-point scheme found in the solver.

This test was then conducted using 1, 2, 4, 8, 16, and 32 processors to obtain calculated potentials. The normalized differences similar to those given in equation *(3.27)* are plotted in Figure 3-8. An unexpected trend was uncovered. The errors incurred through calculation were actually reduced as the number of processors was increased. In fact, the maximum $\Delta$ was reduced from 1.77E-15 in the single processor case to just 1.09E-15 in the 32 processor case. The trend can clearly be seen in Figure 3-8 where fewer and smaller patches of color can be seen in each successive plot.

The reason for this decreasing error is not completely understood. One observation that may shed some light on the matter, however, is that the parallel solver in

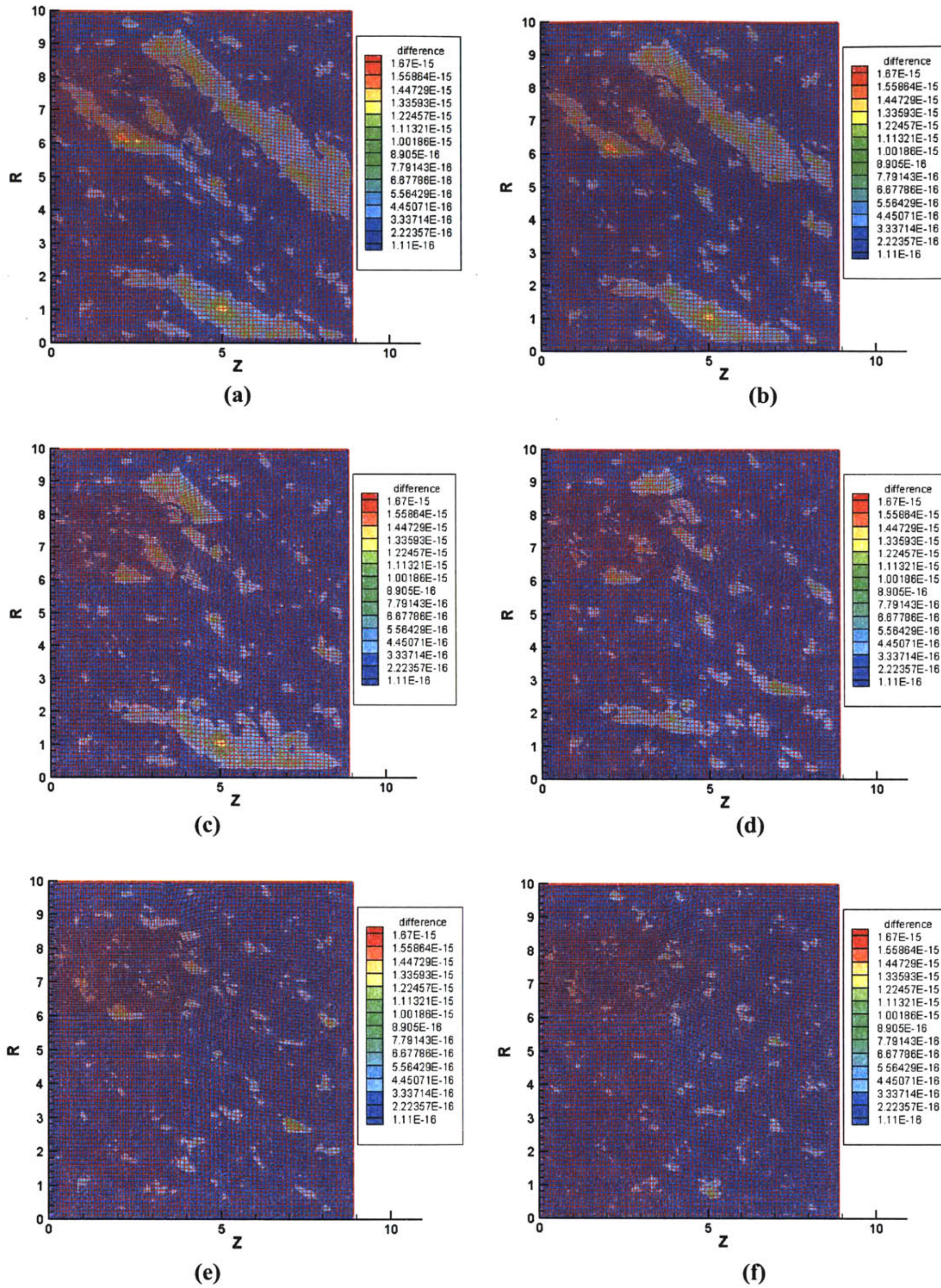



**Figure 3-8: The normalized difference between the calculated and analytic potentials for the Gauss's Law Solver running on (a) 1 (b) 2 (c) 4 (d) 8 (e) 16 and (f) 32 processors. All plots are on the same scale.**

general requires a few extra SOR iterations to converge. This is due, as mentioned previously, to the slightly detrimental red-black ordering necessary for efficient parallel operation. Therefore, it stands to reason that these points would move closer to the analytic solution and attain a higher level of accuracy during these extra few iterations.

## 3.9 Convergence of Successive Over-Relaxation

As we have seen, the successive over-relaxation equation to calculate the potential at point (i,j) and timestep k+1 is given by:

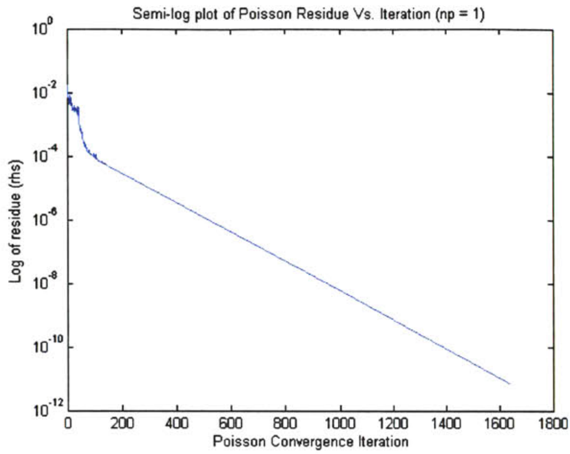
$$\phi_{ij}^{k+1} = \phi_{ij}^{k} + \omega\left(\phi_{ij}^{k+.5} - \phi_{ij}^{k}\right) \tag{3.28}$$

Obviously the iterations therefore converge when the difference $|\phi_{ij}^{k+.5} - \phi_{ij}^{k}|$ tends to some user-defined epsilon. We thus define a normalized value of this residue as:

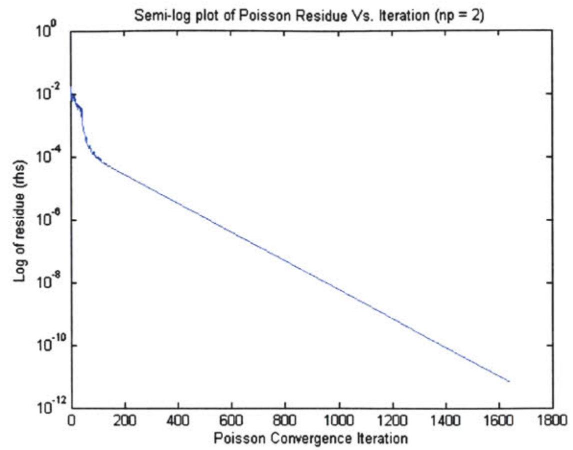
$$RHS = \frac{\left(\phi_{ij}^{k+1} - \phi_{ij}^{k+.5}\right)}{\left(N + S + E + W\right)^{-1}} = \frac{(\omega - 1)\left(\phi_{ij}^{k+.5} - \phi_{ij}^{k}\right)}{\left(N + S + E + W\right)^{-1}} \tag{3.29}$$

In order to judge the convergence rate of the Gauss's Law Solver then, the value of the maximum residue over the grid can be plotted versus the over-relaxation iteration. This was done in Figure 3-9 for the original serial code as well as for the parallel code with various numbers of processors and with a constant $\omega$ of 1.941.

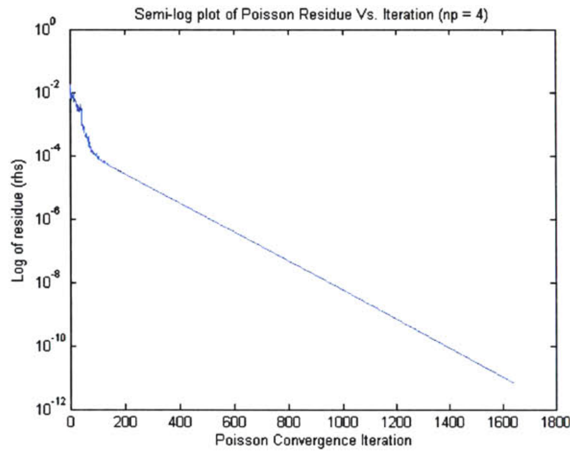
The y-axis of the plots is logarithmic and, after some initial random fluctuations, all 6 plots are seen to be linear, implying that the convergence of the over-relaxation is exponential as expected. Though the plots appear very similar, careful inspection finds that the number of iterations required to reach epsilon (in this case $10^{-11}$) is slightly greater for the parallel cases. As the number of processors is increased, the number of



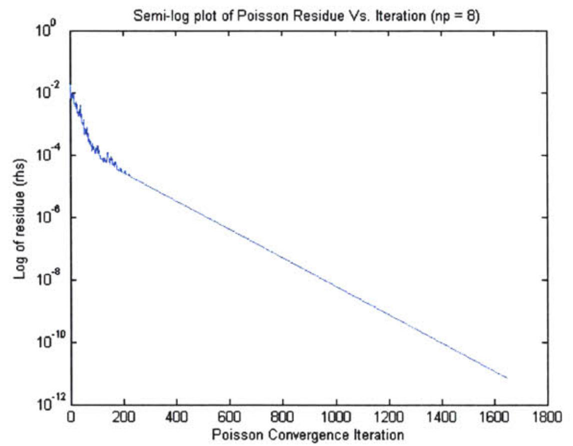

(a)



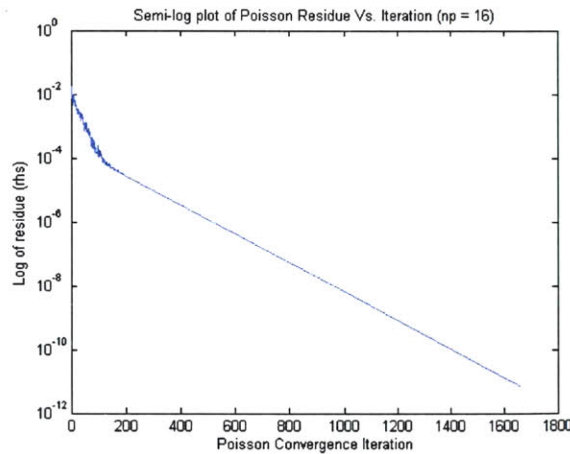

(b)



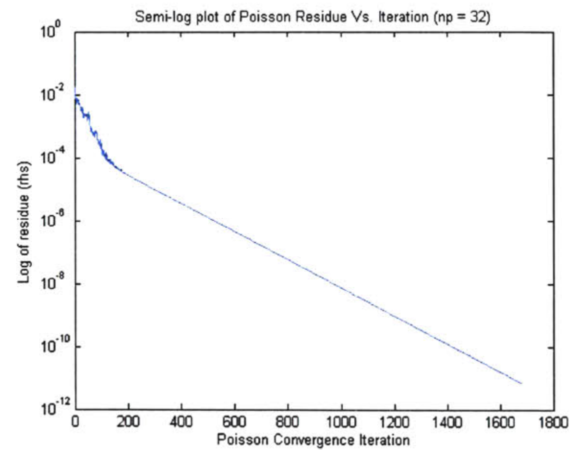

(c)




(d)




(e)




(f)

**Figure 3-9: Plots of the residue RHS vs. over-relaxation iteration for (a) 1 (b) 2 (c) 4 (d) 8 (e) 16 (f) 32 processors. Convergence is exponential, but slightly slower for larger numbers of processors.**

iterations to convergence also increases along with the slope of the convergence. This was the expected detrimental effect of the red-black algorithm employed in parallelizing the solver. Due to the fact that the "black" points use non-updated data in their calculations, the convergence rate is slightly less. It was expected, however, that this slight difference in convergence rate would be more than balanced by the benefits of parallelization.

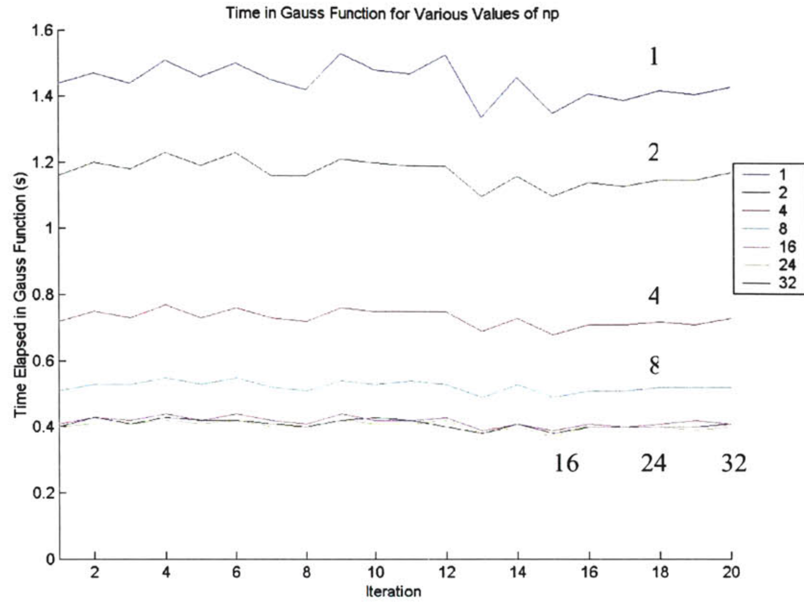
## 3.10 Speed up of the Parallel Solver

More interesting perhaps than the simple fact that the parallelized algorithm converges to the proper result is of course how much speed up was achieved. Speed up is a measure of the utilization of parallel resources and is simply defined as:

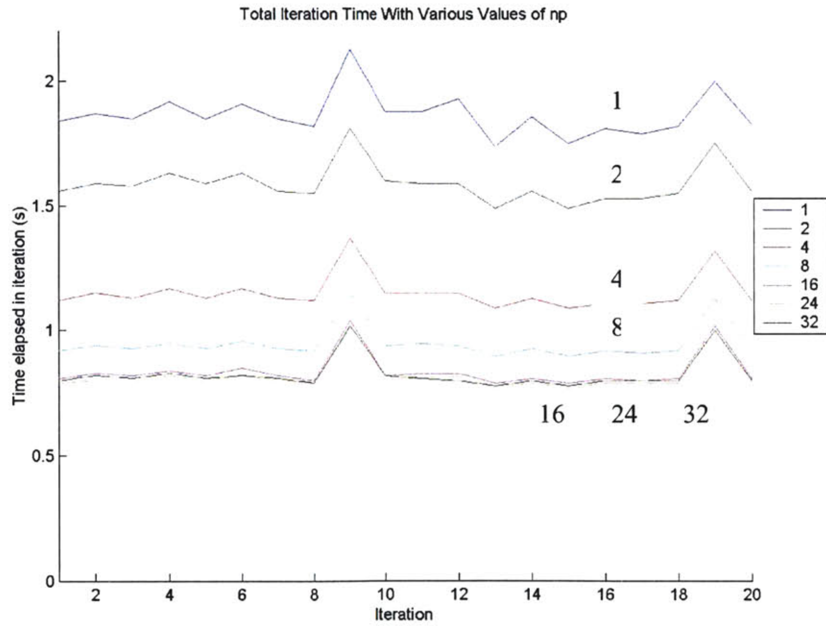
$$\tau_p = \frac{\text{time for 1 processor to complete task}}{\text{time for p processors to complete task}} \qquad (3.30)$$

A related metric is known as the parallel efficiency and is simply defined as $1/\tau_p$. This is a simpler metric for plotting since its range is limited between 0 and 1.

Preliminary speed up measurements were obtained by timing the simulation for 20 iterations with the serial Gauss's Law function and for 20 iterations with the parallel one using various numbers of processors. This initial test was conducted on the 88 by 96 grid shown in Figure 1-2(b). The limited grid size allowed us to first study the parallelization effects using a small number of processors. Figure 3-10 shows the raw time results for these trials, both the overall iteration times and the times spent just in the Gauss's Law function. These data strongly support the belief that the Gauss's Law function is the bottleneck for the code. In fact, it would appear that approximately 80%




**(a)**



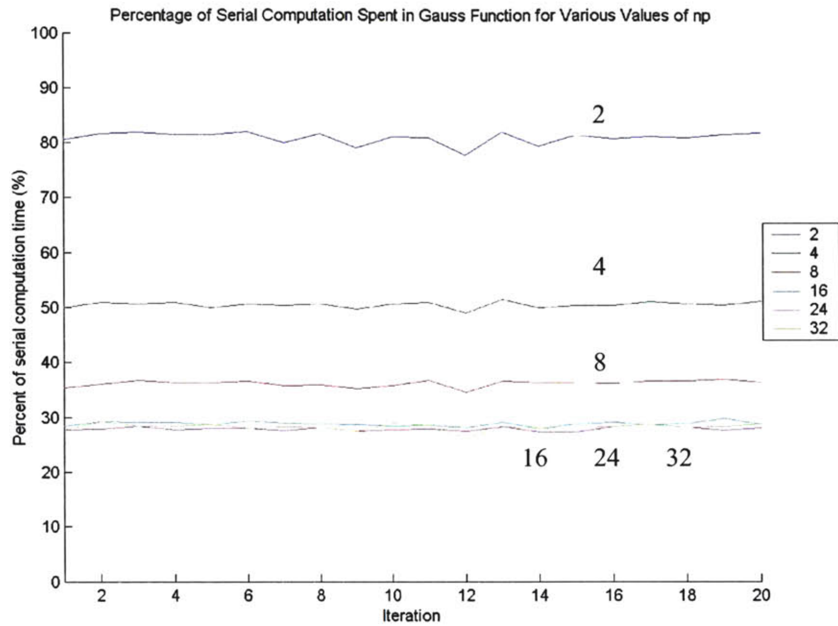

**(b)**

**Figure 3-10: Time spent in (a) Gauss Function and (b) total iteration for various numbers of processors. These data give credence to the belief that the Gauss function is requiring the greatest amount of total iteration time. Note the peaks at iterations 9 and 19. These occur because every 10 iterations, moments are calculated and information is saved.**

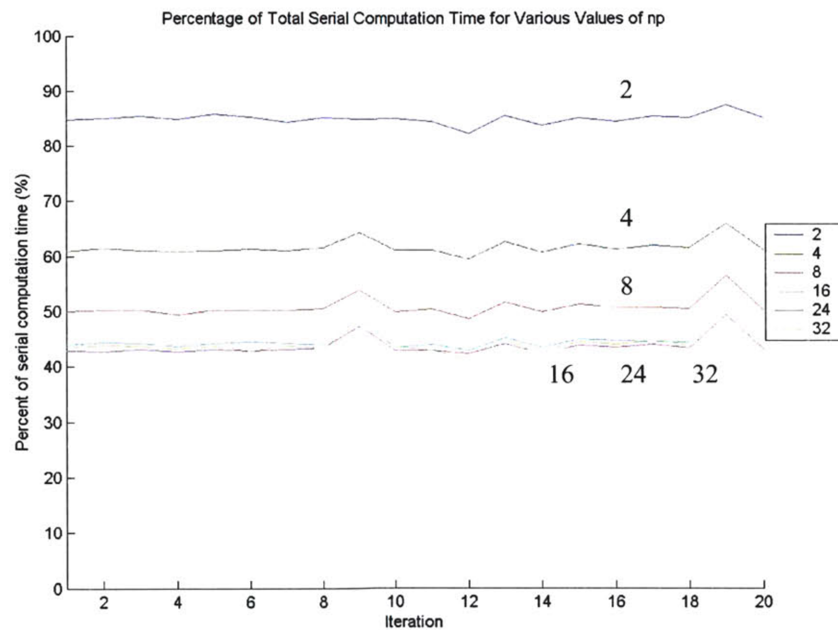
of the computation time is spent in this function. As will be discussed later, these results are unfortunately misleading in that respect and although the speed of the Gauss's Law function was greatly enhanced through parallelization, a similar decrease in total run time was not in fact detected.

Figure 3-11 shows the parallel times as percentages of the serial time. As expected there is a signficant reduction in Gauss's Law solution time and only a slightly lesser one in total iteration time. It might be noted that the percentages for 16 or more processors are clustered in approximately the same region. It seems that given these trial conditions, 16 processors represents the point where the reduced convergence rate and increased communication times tend to balance the increased speed gained by parallelization, and a minimum solution time is reached.

This concept is perhaps better illustrated by Figure 3-12 which shows the average speed up of the parallel code given differing numbers of processors. The graph is clearly sub-linear, and in fact appears to represent an inverted parabola with a maximum at approximately 24 processors. This result was slightly dismaying. Of course, the ideal would be for the speed up to increase linearly with additional processors, and while the diminished convergence rate of the over-relaxation algorithm made it clear that we would certainly not be able to achieve this ideal, a slightly higher maximum had been predicted and desired. The next section of the thesis examines this issue further.

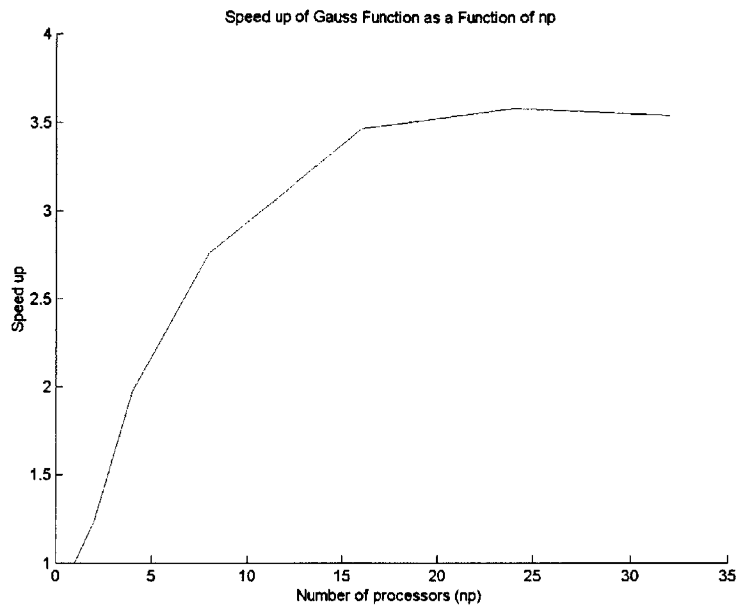



**(a)**

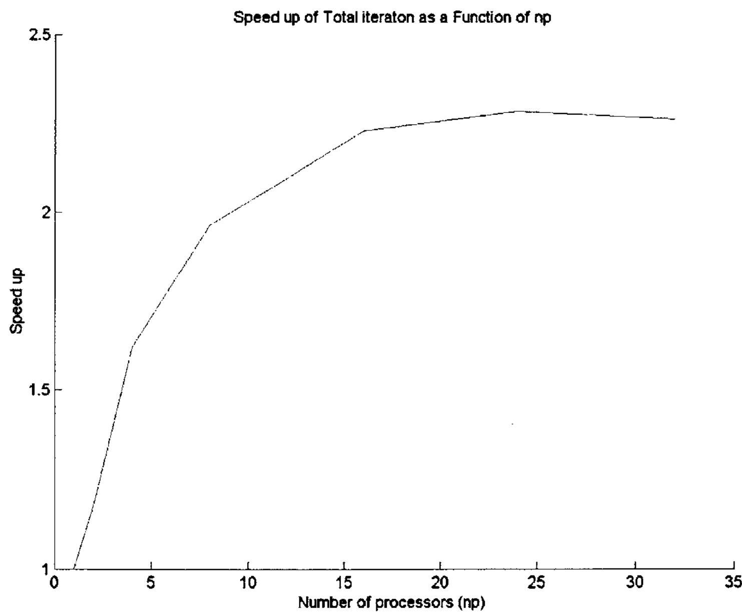



**(b)**

**Figure 3-11: Percentages of serial computation time spent in (a) Gauss function and a (b) total iteration. Adding processor after about 16 does not appear to obtain a significant savings. This may not be the case on a larger problem.**




(a)




(b)

**Figure 3-12: Speed ups of (a) Gauss function alone and (b) the total simulation iteration. Clearly, the algorithm does not achieve the optimal linear speed up ratio. This is likely due to the decreasing rate of convergence of the SOR algorithm as processors are added. Also, as processors are added, the broadcast communication time may become significant. Overall, however, there is a significant reduction in the time required for a simulation iteration.**

## 3.11 Speed up of the Parallel Gauss's Law Solver on a Larger Grid

The sub-linear speed up results for the parallel solver were at first confusing and disheartening. However, it was hypothesized that the relatively small 88 by 96 grid we were testing on was in fact too small to display the full benefits of parallelization. That is, with such a small grid, the time required to calculate the potentials in a slightly larger region actually was turning out to be shorter than the time required both to transmit the boundary data between processors and to perform extra iterations due to the slower convergence rate of the red-black algorithm. However, a quick calculation reveals that the ratio of boundary nodes to inner nodes behaves as the square root of *np*/(NZ*NR). Therefore, as the grid size was increased these communication and convergence times should become negligible when compared with the savings achieved through parallelization.

For this reason, trials similar to those in the previous section were conducted on a much finer grid. This 871 by 951 grid is simply a ten times finer version of the one used in the earlier parts of the thesis. Unfortunately, it is so fine, that when plotted it appears simply as a filled rectangle, and so a useful image of this grid was unattainable.

Once again, the speed up of the Gauss's Law solver was calculated for the various numbers of processors and the result is shown in Figure 3-13. This was the result we had expected. The speed up is extremely linear over the range shown in the figure, indicating that at this larger mesh size, the parallel resources were being used effectively. This is extremely promising for the prospect of later work which might create a fully three-dimensional version of this simulation since the algorithm is well-suited for the large computational requirements such a model would entail. The required number of nodes

for a three-dimensional grid is expected to be about two orders of magnitude larger than for the current two-dimensional simulations.

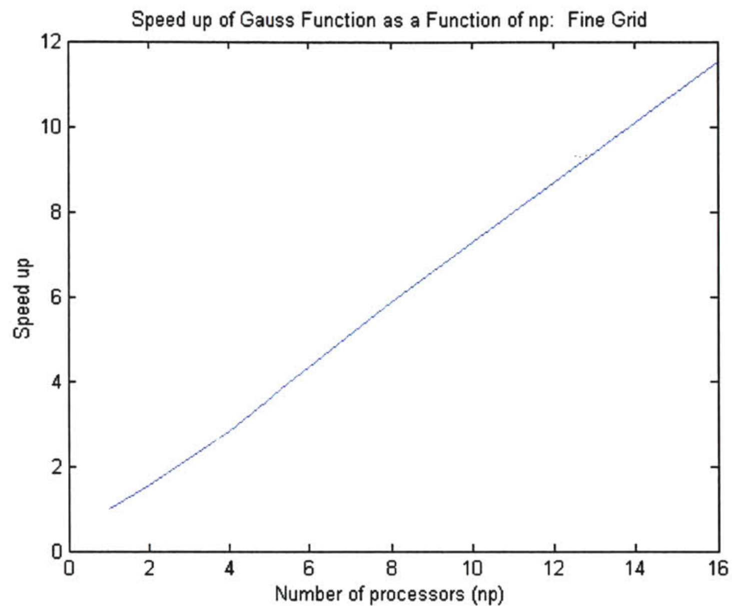



**Figure 3-13: Speed up of the Gauss's Law Function on a ten times finer grid.**

# Chapter 4

# Parallelization of Particle Mover and Collisions

## 4.1 Parallelization of Particle Distribution Creation

The original algorithm described in [26] implemented the creation of initial uniformly random in space particle distributions to be used as the starting point of the simulation. This "seeding" of particles was accomplished by first calculating the volume fractions of each grid cell. Together, these fractions can be viewed as a probability distribution such that:

$$f(i,j) = \frac{V_{i,j}}{V_{total}} \qquad (4.1)$$
$$\sum_{i,j} f = 1$$

An imaginary box is then drawn around the distribution with the box height being equal to the maximum of $f$ over all of the cells. Next, a particular cell (i, j) is selected via two random numbers ranging uniformly between 0 and $N_z$-1 and between 0 and $N_r$-1,

respectively. Finally, a third random number, p, ranging between 0 and the maximum of $f$ is created. If $p <= f(i, j)$ then the particle is placed in cell (i, j). However, if $p > f(i, j)$ then the particle is not placed, and the process is attempted again with a new random position. This algorithm was first proposed by Von Neumann. It is easy to see that in the limit as the number of particles becomes large compared to the number of cells, this technique should produce a practically uniform density of particles per cell.

The parallelization of this algorithm is incredibly transparent. If we desire a total of N particles to be seeded, we simply allow each processor to seed N/*np* of those particles. If N mod *np* is not 0, we of course distribute the particles in the obvious way, such that:

$$\begin{aligned} pid < N \bmod np &\Rightarrow \{N_{np} = \lfloor N/np \rfloor + 1\} \\ pid \geq N \bmod np &\Rightarrow \{N_{np} = \lfloor N/np \rfloor\} \end{aligned} \qquad (4.2)$$

Since the placement of a particle does not depend on where preceding particles have been placed, the parallel procedure should produce the same desired results.

The only caveat which should be noted regards the generation of the random numbers used in the procedure. These numbers were generated using C pseudo-random number generating functions that rely upon a user-defined argument known as a seed. From this seed, the "random" output of the function is generated through a complicated formula [33]. The important thing to note is that given the same seed, this formula will always produce the same sequence of results. Therefore, if each processor was given the same random seed, they would each place their particles in exactly the same places on the grid. In the limit where N/*np* is inifinitely large compared to the number of grid cells, this should not cause a significant problem. If the distribution was assumed uniform for one processor, the sum of the distributions would still be uniform. However, if this limit

is not completely satisfied (which in reality it never is), using the same random seed for every processor will tend to compound the non-randomness of the distribution. That is, if the distribution is non-uniform by some small amount $\varepsilon$, where $\varepsilon$ is found using some linear metric of deviation such as the standard deviation, the sum of the distributions will be non-uniform by $np*\varepsilon$, an amount which could grow significant if $np$ is large.

To avoid this potential problem, different random seeds were used for each processor. Of course, this then presents another reason that the results of a parallel simulation will not exactly match those of a serial one; their initial distributions will not match particle for particle, but will merely be statistically similar.

## 4.2 Parallelization of Neutral Injection

During the simulation, neutral particles are continuously created at the anode and injected into the cells surrounding that region. Szabo's code [26] used the rejection method to determine the positions of these new neutrals such that their time-averaged density was constant. He calculated the number of neutral super-particles to introduce at each time step from:

$$\frac{dN}{dt} = \frac{\dot{m}}{m_n\left(s_0[size]\right)} \tag{4.3}$$

where [size] represents the number of particles in a super-particle of size 1 and $s_0$ is the initial statistical weight of a single neutral. Von Neumann's method described above was again used to determine the positions of these new neutrals.

Since the rejection method was used, this step is again fairly easy to parallelize. Each processor simply injects $1/np$ of the total number of neutrals needed for this

iteration. To deal with the issue of non-zero remainders of this division, however, an additional layer of complexity was required. If we always placed the extra neutrals onto the processors whose *pid*s' were less than dN/dt mod *np*, the total number of neutrals per processor would quickly become unbalanced (as we experienced firsthand in our first trial). An unbalancing of the number of neutrals per processor not only has negative effects on computational performance by unevenly distributing burden, but could also create a situation in which the number of neutrals was not large with respect to the number of processors and the number of grid points, ruining the statistical accuracy of the model. Therefore, it is necessary to keep track of which processor was the last to inject an extra neutral and shift the honor of being the first to inject an extra neutral after each neutral time step. Thus if we assume we have stored the first processor to inject an extra neutral at a given iteration in a variable *injector*, we have:

$$
\begin{aligned}
& pid < \left(injector + \frac{dN}{dt}\right) \bmod np \Rightarrow \left\{ \frac{dN_{pid}}{dt} = \left\lfloor \frac{dN}{dt} \Big/ np \right\rfloor + 1 \right\} \\
& pid > \left(injector + \frac{dN}{dt}\right) \bmod np \Rightarrow \left\{ \frac{dN_{pid}}{dt} = \left\lfloor \frac{dN}{dt} \Big/ np \right\rfloor \right\} \\
& injector = \left(injector + \frac{dN}{dt}\right) \bmod np
\end{aligned}
\tag{4.4}
$$

In this way the processors should remain relatively balanced with respect to the number of neutrals injected at the anode over time.

## 4.3 Parallelization of Ion-Neutral Charge Exchange

The method used to parallelize the ion-neutral charge exchange events was, as usual, based on the model described by Szabo [26]. Charge exchange frequencies depend only

on the neutral density, known globally by all processors, and the relative velocity of a single ion particle. Thus, each processor can calculate the probability of its own ions undergoing charge exchange independently of the other processors.

The number of these exchange events was recorded individually by each processor during the ion time steps. When a neutral time step arrived, the number of such events was summed among all processors. Then a single processor was chosen to apply the charge exchange events to its neutrals. It was necessary to allow only one processor to apply charge exchange to its neutrals, because as neutrals are ionized, the probability of the next neutral becoming ionized actually grows less. Thus, if all processors were allowed to apply the charge exchange events at the same time using the globally summed number of ion-exchange events, too many neutrals would be ionized. If, on the other hand, the ion-exchange events were not summed among processors, and each processor was allowed to perform charge exchange on its own, too few neutrals would be ionized.

Perhaps an example will make this last point more clear. Suppose we have two bags of colored balls. One bag has 2 red balls and 3 black. The other bag has 1 red ball and 4 black. If we get to pick twice from the first bag and once from the second bag, the probability of getting 3 red balls is (2/5)(1/4)(1/5) which can be written as 1/50. If we instead put all the balls together in one bag, we would have 3 red balls and 7 black. Now if we pick three balls from the bag, our chances of getting 3 red balls would be (3/10)(2/9)(1/8) or 1/120. Clearly the probability distributions are different in each case. This situation is analogous to the parallel and serial ion-neutral charge exchange model in which the probability that a neutral is selected for charge exchange is equal to the number of ions that were charge exchanged divided by the total number of neutrals available for

exchange. If we desire the parallel code to emulate the serial code as closely as possible, one processor should perform all of the charge-exchange events.

## 4.4 Parallelization of Ion-Neutral Scattering

Another type of collision event modeled in the code was ion-neutral scattering. This type of collision was modeled in the serial code by using a hard sphere approximation to exchange energy and momentum between the ions and neutrals. The total amount of energy and momentum given up by the ions during their movement was tallied and saved until a time step in which the neutrals were moved as well. The application was performed by first checking to see if any scattering had occurred in a given computational cell. If scattering had occurred, a single neutral particle was selected from that cell and all of the accumulated momentum and energy change from the scattering of ions was applied to it.

The parallelization of this section proceeded as follows. The momentum and energy change of the ions was tallied individually by each processor, and each processor adjusted its own ions' properties. This could be accomplished because the frequency of ion-neutral scattering is dependent only upon the neutral density which is global to all processors and the relative velocity of a particular ion. The scattering of one ion does not depend on the scattering of another in the given model, and therefore we can easily divide the work of ion-scattering among the various processors.

When a time step arrived on which the neutrals would be moved, the change in neutral energy and momentum was applied individually before the neutrals were moved. This was accomplished by first summing the tallies of ion momentum and energy loss

among all of the processors. Then a single processor performed the actual application of the neutral property change, notifying the remaining processors whether or not there were further changes to still be made at the next available time. This was done for reasons similar to those discussed in Section 4.3. To ensure the velocity distributions of neutral particles remained roughly similar on each processor, the processor that was allowed to apply the neutral-scattering was rotated every neutral time-step in a manner similar to that described in Section 4.1.

## 4.5 Parallelization of Neutral Ionization

The ionization of neutrals by high energy electrons is modeled in [26] using available empirically derived cross-sections. These cross-sections are dependent only upon the electron temperature interpolated to a particular electron's location. With one small inconvenience, the probability of an ionization event occurring can therefore be calculated independently by each individual processor.

The number of ionization events is recorded after each electron time step. On the next neutral time step, the number of such events is summed over all processors and over the preceding electron time steps. As in the case of charge exchange, these sums are given to one particular processor to be applied to its neutrals. Each processor takes a turn at ionizing its neutrals on successive neutral time steps.

The small catch that slightly complicated this process was that in calculating the number of ionization events, the current neutral density must be used. This means that neutrals which have just been ionized should no longer be included in the neutral density. If we perform this calculation completely in parallel, however, one processor would not

know how many neutrals its comrades had ionized and would be unable to properly update its neutral density. To perform the calculation exactly correctly then would undoubtedly require a serial method. Instead, we chose to approximate this correction to the neutral density which is so small as to be almost negligible anyway. Our approximation just assumes that if a particular processor has ionized a neutral, the distribution of particles on the other processors should be roughly equivalent, and so we have the processor assume that all of its fellows have also ionized a neutral. Thus, we decrease the neutral count by the number of neutrals ionized on the current processor times the number of processors. In the limit where the number of particles and grid nodes are large compared to the number of processors and there is a large number of particles per computational cell, this should be an adequate approximation.

## 4.6 Parallelization of Double Ionization

This event, the creation of $Xe^{+2}$ from $Xe^{+1}$ via the impact of a high energy electron, is handled exactly like the single ionization events, without the special caveat of needing to update the particle densities after each event. In theory, we should have had to update the densities after each event since the probability of a double ionization event depends upon the temperature of the initiating electron and the background ion density. However, following Szabo, we claimed that this correction was particularly negligible and did not include it in the calculations.

## 4.7 Parallelization of Cathode Electron Injection

A slightly more challenging section of the code to parallelize was the injection of cathode electrons along the right hand side boundary of the simulation region. The difficulty here was not so much conceptual, considering that the technique used was equivalent to the ones described above. However, settling on an implementation which retained as much balance between the nodes as possible proved slightly tricky.

In Szabo's work, the number of electrons which were injected by the cathode along the right hand side boundary was calculated from the total charge difference existing in the final column of computational cells along with a few other parameters based on the potential and temperature at those points. The latter parameters were kept global among processors, and so they do not need to be further discussed since any or all processors were able to access this information. If the boundary was positively charged, electrons were injected. If the boundary was neutral or negatively charged, nothing was done.

The most parallel method of implementing this portion of the code, and the method that was attempted first, was to maintain the charge densities locally on each processor. Then each processor would simply inject the number of electrons it required to retain neutrality along the right hand side. With a little thought and understanding, however, we come to the conclusion that this method will not yield the same results as the serial model. The problem lies in the fact that electrons are only injected if there is a positive charge. For instance, if seven processors are negative by one charge at the boundary and the eighth is positive by five charges, this method will inject five electrons

into the system. The serial version would instead see a total negative charge of two and inject no electrons.

Instead, we must once again tally the charge differences among all the different processors. Then, all of the processors calculate the number of electrons which need to be injected. This may seem wasteful, but assuming our processors are all of equal speed, it is actually faster computationally than asking just one processor to do the calculation and then sending its results to the other processors. Once every processor knows the total number of cathode electrons which should be injected, a technique very much like the injection of neutrals is used. The total number of cathode electrons is divided as evenly as possible among processors, with the extras being doled out in a rotating manner. This ensures to the maximum extent possible that the total number of electrons on each processor will be nearly equal.

## 4.8 Preliminary Speed up of Particle Mover

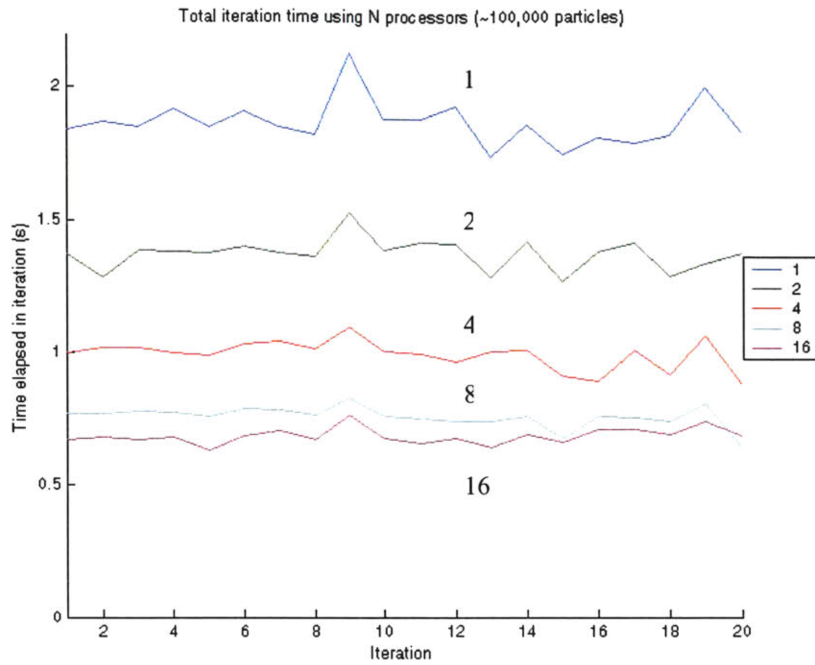
As with the Gauss function alone, measurements of the parallel efficiency of the remainder of the code were obtained. Both the serial and parallel simulations were timed for their first twenty iterations, with the latter utilizing 2, 4, 8, and 16 processors. The raw time results for these tests are shown in Figure 4-1(a).

When a simulation begins, it typically contains approximately 60,000 neutral superparticles, 20,000 electrons, and 20,000 ions. However, as can be seen from Figure 5-1, the electron, ion, and neutral counts can easily reach 350,000 during peak ionization periods. With this in mind, we also timed both the serial and parallel algorithms for

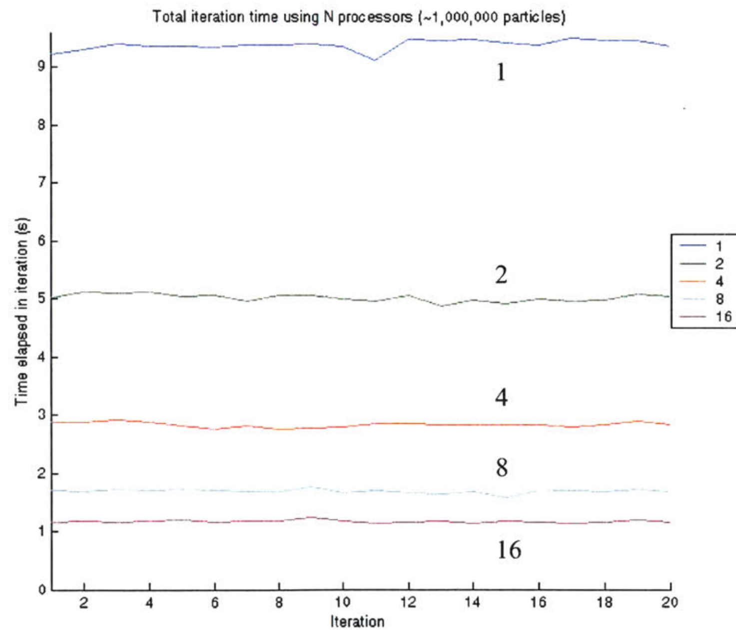
twenty iterations of a simulation containing on the order of 1,000,000 particles. The raw results for these tests can be seen in Figure 4-1(b).

In Figures 4-2(a) and (b) the time required to perform everything in an iteration except the calculation of the potential via the Gauss's Law function is shown. This included the movement of particles, the calculation of particle moments, and the storing and saving of data. Figures 4-3(a) and (b) again show this same information, but normalized to depict the percentage of the serial computation time required by the parallel code for these tasks. Note the chaotic nature of Figure 4-3(a). This indicated a bottleneck in performance. It is due to the structure of the parallel code. Many tasks, such as the computation of temperature across the grid or the storing of data files to disk take at best equal times in the parallel and serial cases. Some may even take longer in parallel as more processors are added due to the increased communication costs and overhead involved. These tasks are generally a small portion of the computation time. However, when there are only a few particles to be moved, the drawbacks of these small tasks tend to dominate the benefits gained through the division of labor in other portions of the code. In Figure 4-3(b), however, the time required to move the particles becomes dominant and a clearly decreasing trend in computation time is plainly seen.

Figures 4-4(a) and 4-4(b) depict the speed ups calculated for these tests. Again, Figure 4-4(a) shows that parallelization was not necessary when only 100,000 particles were present. However, the nearly linear trend obtained in Figure 4-4(b) showed that when the problem size became larger, the parallel resources were being effectively used to cut down the computational time.

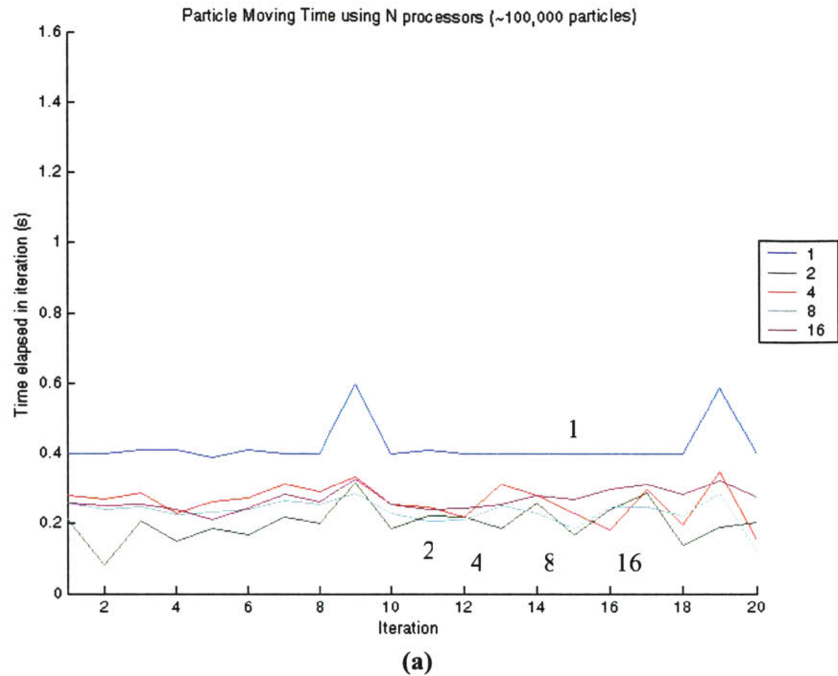



**(a)**

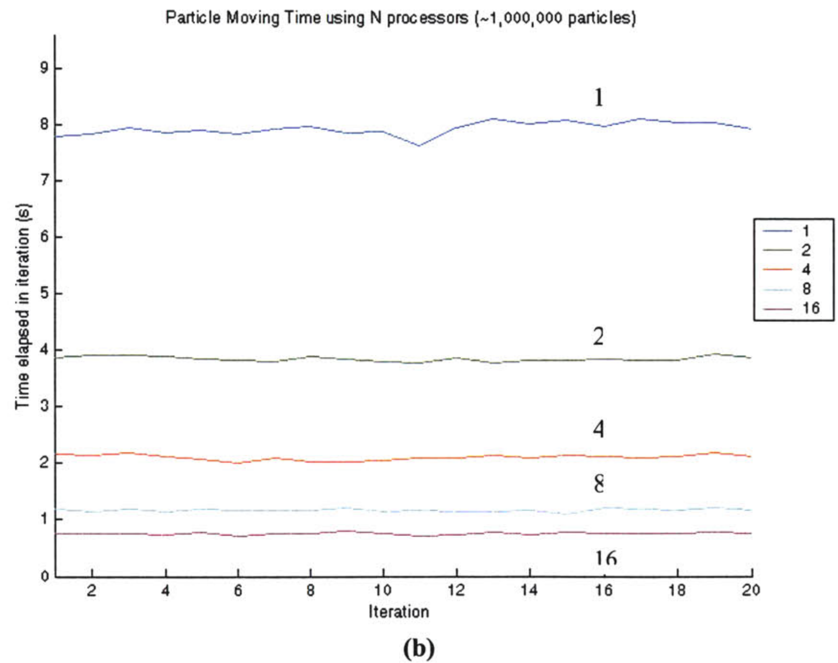



**(b)**

**Figure 4-1: The total time for a single simulation iteration was plotted over the first 20 simulation iterations for the cases of 1, 2, 4, 8, and 16 processors. Figure 4-1(a) shows the times for a simulation having on the order of 100,000 particles while Figure 4-1(b) shows the times for a simulation having on the order of 1,000,000 particles.**

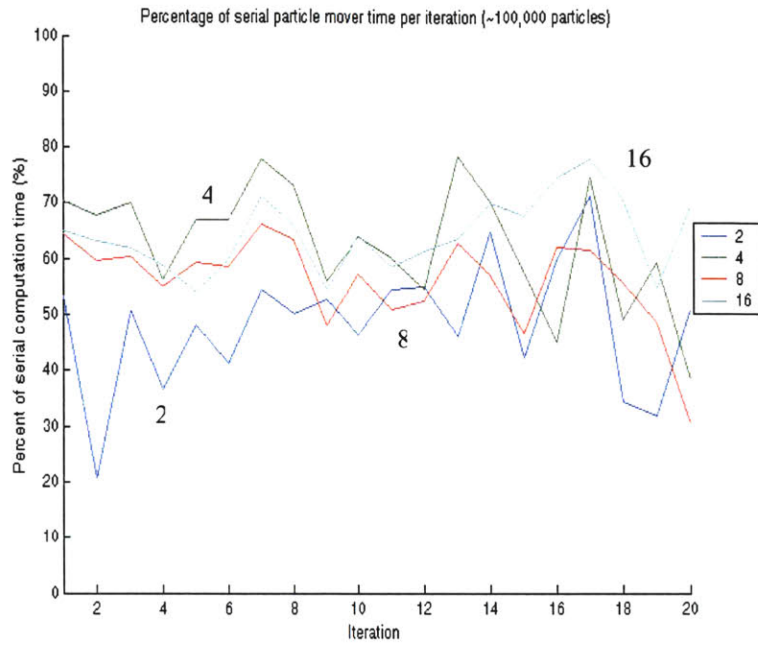



**(a)**

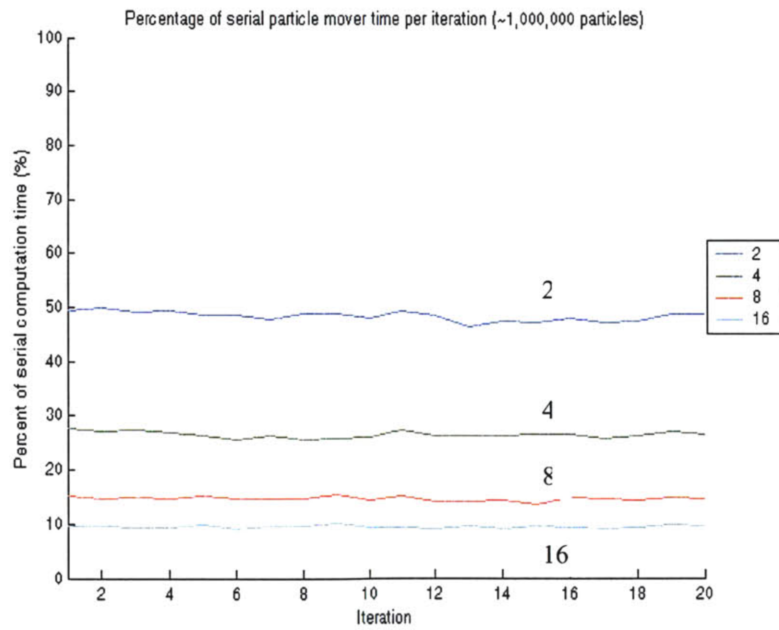



**(b)**

**Figure 4-2: The time required to do everything except calculate the potential was plotted in the figures above for the cases of 1, 2, 4, 8, and 16 processors. Figure 4-2(a) shows data from a simulation containing on the order of 100,000 particles and Figure 4-2(b) shows data from a simulation containing on the order of 1.000,000 particles.**

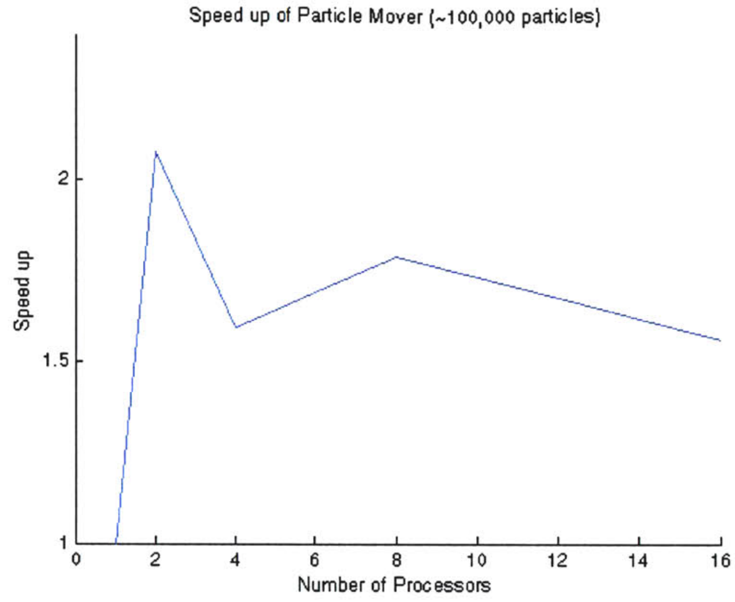



**(a)**

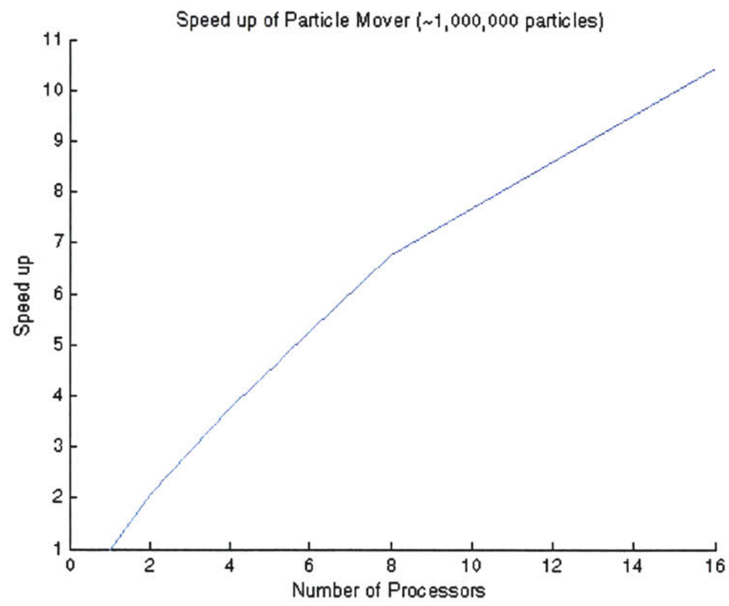



**(b)**

**Figure 4-3: The percentage of serial time required to move (a) 100,000 particles and (b) 1,000,000 particles was plotted for the cases of 2, 4, 8, and 16 processors.**




**(a)**




**(b)**

**Figure 4-4: The speed up of the particle mover portion of the simulation was plotted for (a) a simulation of approximately 100,000 particles and (b) a simulation of approximately 1,000,000 particles**

# Chapter 5

# Parallelization Results

## 5.1 Preliminary Results of the Fully-Parallelized Code

Once the entire code was parallelized, we began conducting trials to ascertain how greatly the alterations had affected the actual results of the simulation. The optimum here, of course, would be for the serial and parallel results to agree exactly and completely. However, we were already certain that this would not be the case for a number of reasons mentioned above. Despite some minor differences, however, we found that the parallel results did overall agree quite well with the serial results qualitatively, if not in exact details.

## 5.2 Preliminary Timing Results

Three trials were conducted on the Compaq Alpha parallel machine using the P5 thruster geometry and the key parameters shown in Table 1-1. The first trial was conducted using only one processor and required a total of 101.72 hours to complete. A

second run using four processors required only 45.32 hours. Finally, an 8 processor run finished in 31.90 hours. As was observed in other preliminary tests, linear speed up is certainly not attained for this problem. The problem is simply too small and the parallelization overhead too large to make completely efficient use of the resources. However, the speed up is still clearly significant, with 8 processors finishing more than 3 times as fast as the serial implementation.

## 5.3 Analysis of Electron Number Data

Figure 5-1 below shows the total number of electrons present in the simulation region plotted over the course of a particular run. The data have been plotted for the tests conducted with 1, 4, and 8 processors.

The first feature of interest is the initial peak which occurs at approximately 32,000 iterations for the serial trial. As can be seen from the figure, the peak arrives slightly sooner, at 31500 and 31000 iterations, for the 4 processor and 8 processor cases, respectively. This 1.5% and 3% difference in arrival time does not significantly affect the conceptual conclusions of the results, and was originally believed to be due to the change in the random seed of the simulation. As was later discovered and explained in Section 5.6, the difference can in fact be attributed to the instability of the SOR algorithm.

Another, more disturbing feature of the results is the secondary peak which occurred at about 139,000 iterations in the serial case, but arrived at approximately 120,000 and 119,000 iterations in the 4 and 8 processor cases, respectively. The

remainder of the electron number data clearly agrees between trials to within a few percent.

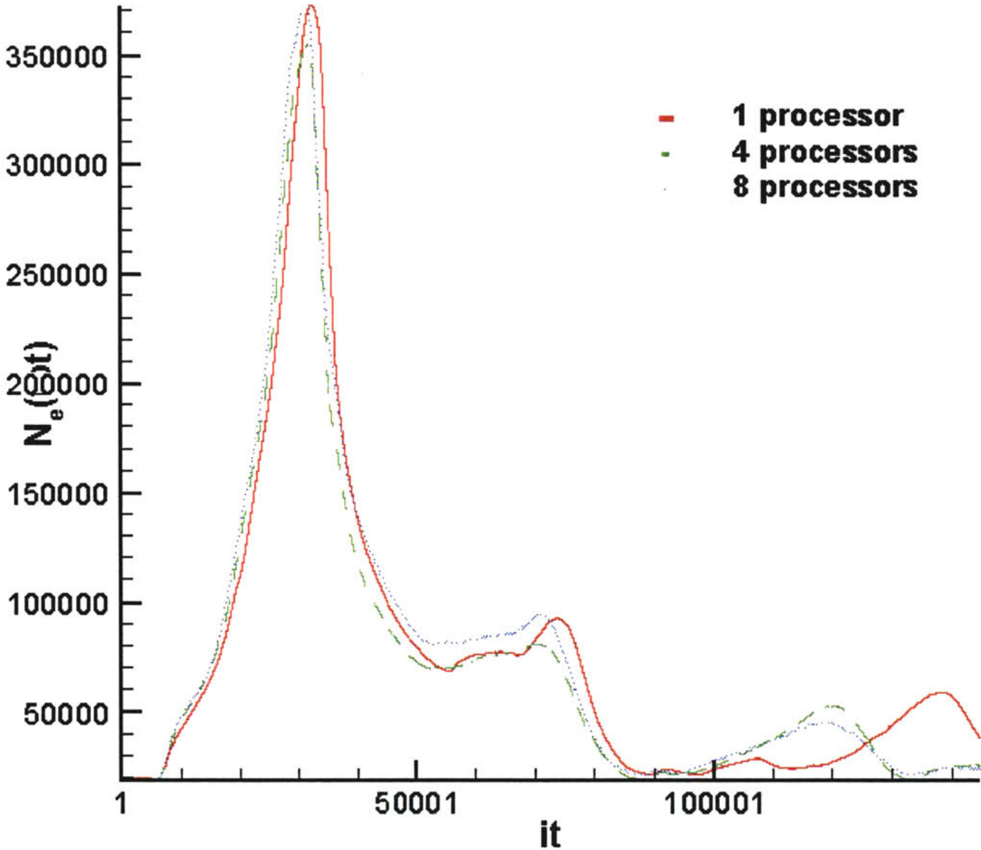



**Figure 5-1: Number of electrons versus simulation iteration for various parallel cases.**

## 5.4 Analysis of Neutral Number Data

The number of neutrals in the simulation is plotted below in Figure 5-2 over a full trial of 145,000 iterations for the cases of 1, 4, and 8 processors.

These results match one another very closely. We can see the expected neutral dip around 30,000 iterations where the peak of ionization occurred. The neutral distribution then recovers itself and slightly overcompensates, before asymptoting down

toward a steady state value around 100,000 iterations. The parallel and serial results here agree to within a few percent of one another at every point, and as can be easily seen from the chart, there are no systematic differences. For instance, the 8 processor case actually achieves a slightly higher maximum neutral number than the serial case while the 4 processor case achieves a slightly lower maximum. These differences again are attributable to the results discussed in Section 5.6.

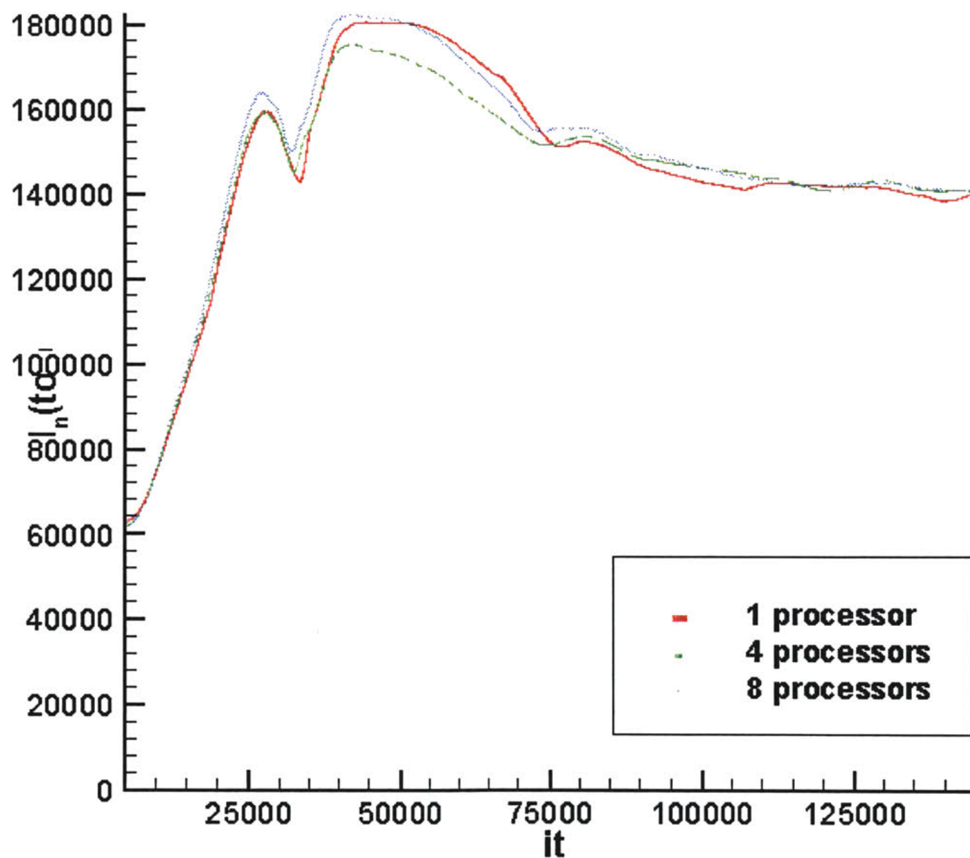



**Figure 5-2: Number of neutrals versus simulation iteration for various parallel cases**

## 5.5 Analysis of Anode Current Results

Plotted side by side in Figure 5-3 below are the anode currents at each iteration for the cases of the three different numbers of processors.

The main peak magnitudes are all similar, with the serial trial attaining approximately 90 Amps and the parallel trials reaching about 88 and 89 Amps, respectively. The secondary peak magnitudes that occur between 70,000 and 75,000 iterations in all cases were also very similar with the serial and 8 processor trials reaching approximately 24 Amps and the 4 processor case reaching about 22 Amps.

As with the numbers of electrons we see that the final peak of the current shown in Figure 5-3 occurred earlier in the parallel cases as compared to the serial case. This is actually a quite encouraging sign. If the results for current did not agree with the structure of the results for electron number, it would clearly indicate a severe problem in the simulation. However, as would be expected, the anode current correlates strongly with the number of electrons present, and this dependency was accurately portrayed in both the serial and parallel cases.

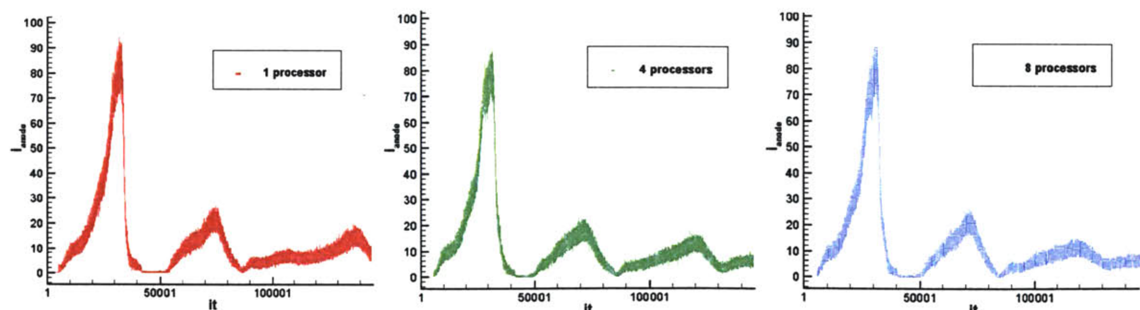



**Figure 5-3: Anode Current Versus simulation iteration for various parallel cases.**

## 5.6 Fixing Non-Convergence

The sizable qualitative differences between the serial and the parallel results in the preliminary experiments described above were dismaying. The parallelization did include some approximations which would imply that it would not obtain exactly particle for particle the same results as the serial model, but we had hoped and theorized that these differences would be small and grow smaller as the problem size was increased and the statistical limit was reached. With this trend not being observed in a series of runs with progressively scaled-up numbers of particles, we began to wonder if there might be some bug in our work and laboriously began testing each section of the code.

During the course of our investigations, we discovered that the SOR Gauss's Law solver handed to us was not necessarily stable over all iterations, and in fact non-convergence of the electric potential was seen in both the serial and parallel trials. The over-relaxation factor, $\omega_{opt}$, used in the SOR solution process had been set to 1.941 by one of the many hands which the MIT PIC code passed through on its way to us. Szabo claimed that his thesis trials were stably convergent using an $\omega$ of 1.96, but that a "nicer convergence" was obtained if an $\omega$ of 1.918 was used [26]. Of course, Szabo was modeling a mini-TAL thruster and not a P5; the size of the grid plays an important role in the choice of the over-relaxation factor. Blateau then must have set the $\omega$ to 1.941 which appeared to be stable for the majority of iterations on his newly-created P5 grids characterized by larger numbers of larger grid cells. However, we discovered that around 70,000 simulation iterations into the simulation, the Gauss's Law solver became unstable and did not reach the desired level of convergence within the allotted maximum number of iterations. It is our hypothesis that this previously overseen problem remained

undetected since it was only noticeable during the running of the code as an increase in the time required for these non-convergent iterations, and since it was in the middle of the simulation, there was no reason to monitor this portion of the trial very closely.

The effect of even a single non-convergent iteration is not negligible. To demonstrate this, a charge distribution which caused the code not to converge was found. The parallel code with four processors was then allowed to run for the maximum 50,000 relaxation iterations at an over-relaxation factor $\omega$ equal to 1.941. The electric potential and fields calculated from this charge distribution were recorded. Next the same charge distribution was used by the four-processor parallel code with an over-relaxation factor of 1.800. Such a low value of $\omega$ was used to ensure that convergence would certainly be reached, and indeed, convergence to a precision of $10^{-11}$ was achieved after almost 10,000 iterations. The absolute differences between the calculated potentials and fields in these two cases are shown in Figure 5-4.

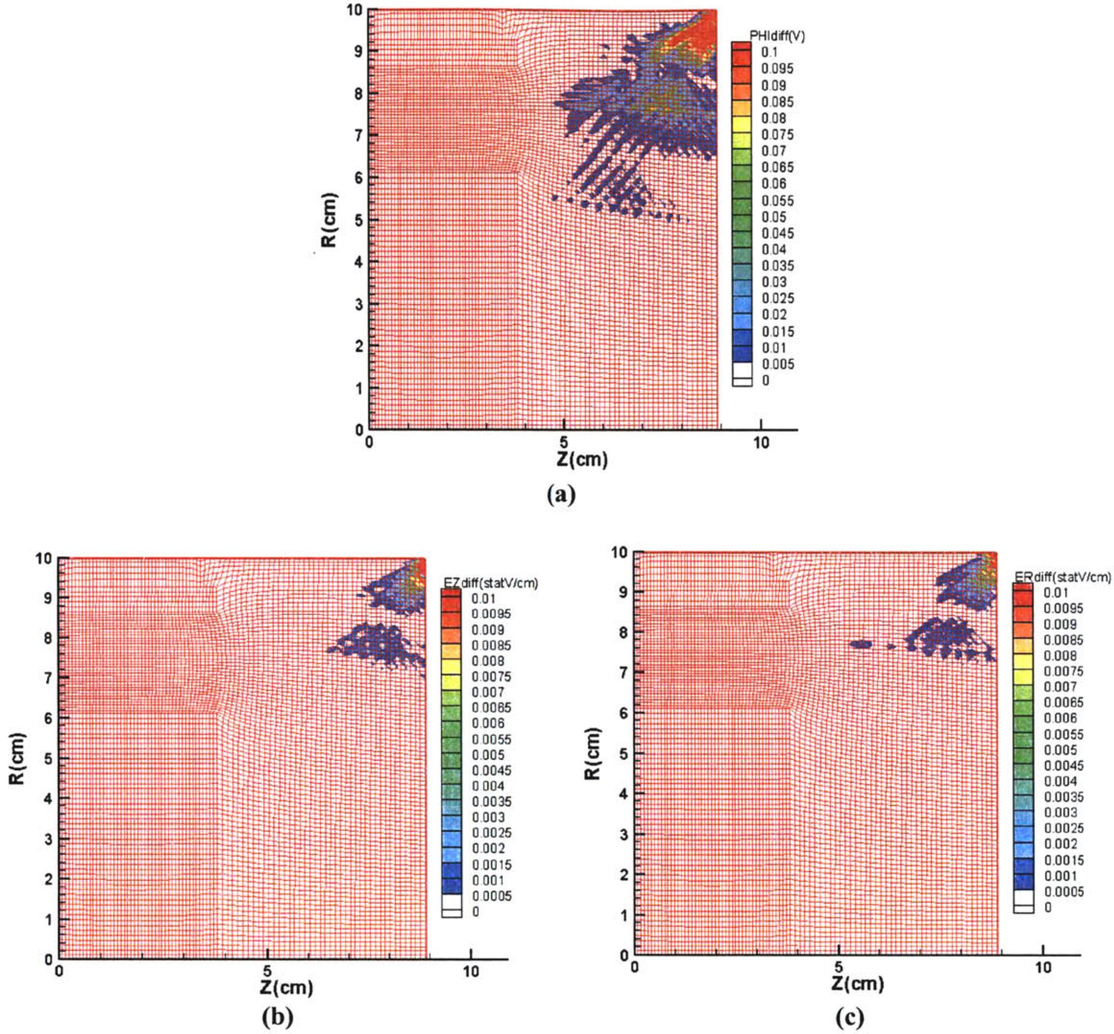



**Figure 5-4: The absolute differences between the (a) electric potential, (b) Z electric field, and (c) R electric field for a converged SOR iteration versus a non-converged iteration.**

The maximum differences are observed in the upper-right corner of the solution region whose gradient boundary condition is apparently responsible for the difficulty in convergence. However, even in the thruster channel there are differences between the electric fields of the converged and non-converged cases which are on the order of $10^{-6}$, orders of magnitude too large for our desired $10^{-12}$ tolerance. Over time, these seemingly small differences can alter the results of the simulation significantly.

Figures 5-5 and 5-6 below show several plots of the number of electron superparticles in the simulation versus the normalized simulation time, T, where 1T is approximately 7 ns. In particular, Figure 5-5(a) depicts results from the serial code as it was originally given to us. The SOR over-relaxation factor, $\omega$, had been set to 1.941. In addition, 10,000 had been set as the maximum number of relaxation iterations available. After this number of iterations, if the algorithm had still not reached the desired accuracy of convergence, the last-calculated electric potential was taken to be correct and the code continued. In Figure 5-5(b) is shown the results of a similar serial trial except that the maximum number of relaxation iterations had been increased to 50,000. If the Gauss Solver had been converging to the desired level of accuracy within the original allotted maximum, these two plots should have been the same. The differences that begin to become extreme around 6000T simulation time show how drastic an effect the early-stopping of the SOR solution was having on results.

With this evidence in mind, we began to investigate why the Gauss's Law solver was not converging and more importantly how we could make it converge for every iteration. From earlier experience, it was known that the Gauss's Law solver sometimes became unstable and diverged with omega's larger than 1.941, and so we wondered if perhaps this over-relaxation factor was on the border of stability. That is, at early iterations when the solution for the potential was simple, the solver might have been stable, but at later iterations, the solver with $\omega$=1.941 might actually have become unstable. Thus, in Figure 5-5(c) is pictured a serial trial of Szabo's code where again the maximum number of over-relaxation iterations has been set to 50,000 iterations, but now

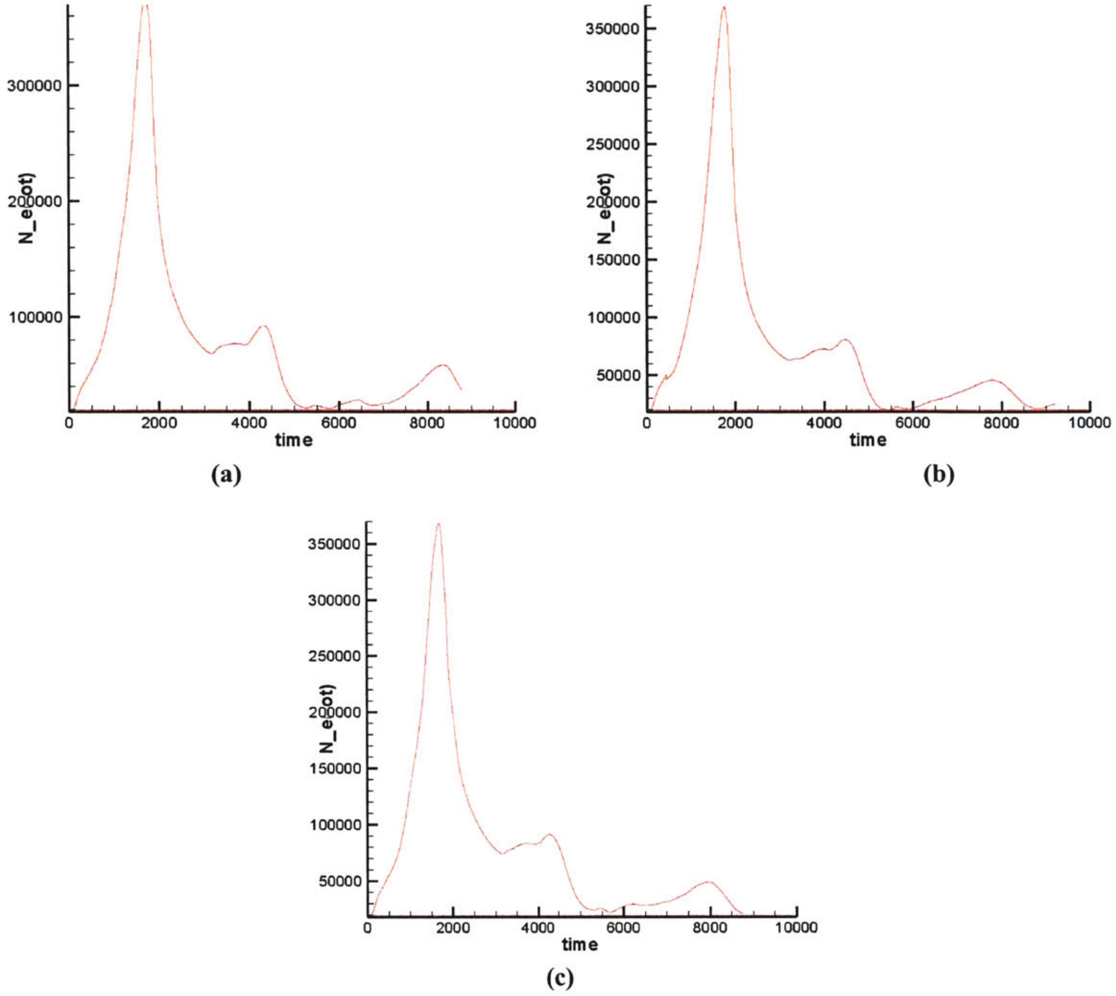



**Figure 5-5: Depicting the electron number versus normalized simulation time (1T ~ 7E-9s) for three different serial experiments with varying Gauss's Law over-relaxation parameters.**

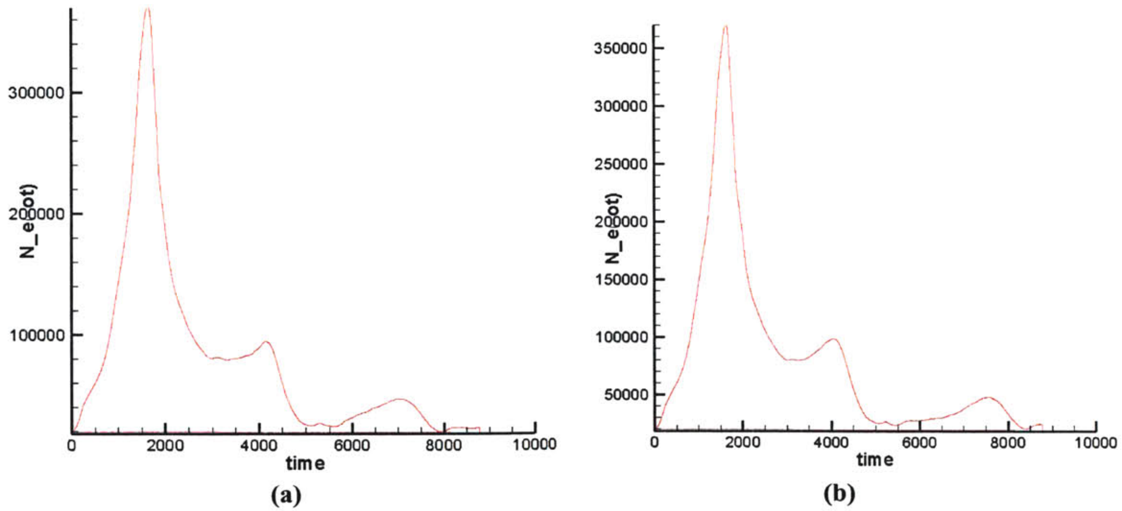



**Figure 5-6: Depicting the electron number versus normalized simulation time (1T ~ 7E-9s) for two different parallel experiments with varying Gauss's Law over-relaxation parameters.**

the ω factor had been reset to 1.918, the number that Szabo had suggested for his mini-TAL thruster. The reader will note the slight differences between Figures 5-5(b) and Figures 5-5(c), especially in the region around 7000T and later. If the Gauss's Law solver had been converging properly at the omega of 1.941, there should have been zero noticeable difference between these two runs except, of course, their time to completion.

This meant that the serial code which we had been attempting to match was not itself yielding accurate results, so we were trying to match incorrect data. As such, it was necessary to adjust the Gauss's Law solver, as will be discussed in Section 5-7, to ensure convergence of the electric potential. Unfortunately, the change in the serial code more or less invalidated any comparisons we had hoped to draw between our parallel results and the results of Blateau [5] and Sullivan [25] who did not report noticing such a non-convergence issue.

The effects of the SOR non-convergence could also, of course, be noticed in the case of the parallel trials. Experiments similar to the serial trials mentioned above were conducted to discern the effects of better convergence using the parallel code and 4 processors on the Compaq Alpha to discern the effects of better convergence. The first of these trials, shown in Figure 5-6(a), kept omega equal to 1.941, but increased the maximum number of possible SOR iterations to 50,000. There were still major differences between this parallel trial and the comparable serial trial shown in Figure 5-6(b). However, when ω was decreased to 1.918 and the SOR iteration maximum was again set to 50,000 iterations, Figure 5-6(b) shows that the parallel and the serial results finally seem to begin to agree. The most important comparison to make is between Figures 5-5(c) and 5-6(b), which are the closest to accurately converged and similar in

nature except that 5-6(b) is a parallel trial and 5-5(c) is serial. It is seen that the smaller electron number peak at about 8000T still comes slightly sooner in the serial case than in the parallel, but the difference has now been reduced to only about 200T. This residual difference exists because none of the experiments shown here were able to completely eradicate the non-convergence problem. There were still a number of iterations, on the order of 500, which reached the maximum number of SOR iterations allotted without converging, and since the parallel solver converges slightly slower than the serial solver, this is a cause for differences between the two results.

## 5.7 Addition of Chebyshev Acceleration to SOR

Judging that the non-convergence of SOR was either due to an overly-sanguine estimate of $\omega_{opt}$ or at least was aggravated by it, we were forced to decrease this parameter during the latter stages of the simulation, that is after approximately 70,000 simulation iterations. Of course, this decreased $\omega$ factor would have meant an increase in the total number of over-relaxation iterations required. In order to balance this, we decided to implement a technique known as Chebyshev acceleration [21].

The principle behind this method is to realize that while a particular choice of $\omega$ may be optimal in the sense of asymptotic convergence, it will not in general be the optimal choice at every relaxation iteration. By altering the $\omega$ at early over-relaxation time steps we can increase the rate of convergence. One well-known choice of $\omega$'s for a typical red-black ordering scheme like ours relies on the spectral radius of the Jacobi iteration:

$$\rho_{Jacobi} = \frac{\cos\frac{\pi}{N_z} + \left(\frac{\Delta Z}{\Delta R}\right)^2 \cos\frac{\pi}{N_R}}{1 + \left(\frac{\Delta Z}{\Delta R}\right)^2} \tag{5.1}$$

where $\Delta Z$ and $\Delta R$ are of course the grid node separation in the respective directions. Unfortunately, there exist certain reservations on whether the classical $\rho_{Jacobi}$ for a uniform rectangular domain are directly applicable to a non-uniform mesh like ours. Therefore, our use of the spectral radius in what follows must remain only an approximation.

From the spectral radius, the optimal $\omega$ at each half-sweep is calculated via the formula:

$$\begin{aligned} &\omega^{(0)} = 1 \\ &\omega^{(1/2)} = 1/(1 - \rho_{Jacobi}^2/2) \\ &\omega^{(n+1/2)} = 1/(1 - \rho_{Jacobi}^2 \omega^{(n)}/4) \end{aligned} \tag{5.2}$$

This recursion tends to the limit:

$$\omega^{(\infty)} \to \omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho_{Jacobi}^2}} \tag{5.3}$$

This limit is the $\omega$ which provides the best asymptotic rate of convergence.

In our non-linear case, $\Delta Z$ and $\Delta R$ are not constant across the entire mesh. Therefore we could have optimized $\rho_{Jacobi}$ at each grid point and ended up with a different over-relaxation parameter, $\omega$, at each point. However, the increased computation to calculate the spectral radius at each point seemed burdensome and unnecessary. Additionally, we already knew the asymptotic $\omega_{opt}$ which worked best for our particular problem. Therefore, we chose to back-calculate the spectral radius from the given $\omega_{opt}$ we wished to attain. That is:

$$\rho_{Jacobi} = \sqrt{1-\left(\frac{2}{\omega}-1\right)^2} \qquad (5.4)$$

The final algorithm then began with a $\rho_{Jacobi}$ which yielded an $\omega$ of 1.941. Armed with this $\rho_{Jacobi}$, we could then use equation *(5.2)* to calculate an $\omega$ at each over-relaxation iteration. Thus, a different $\omega$ is being used for each individual relaxation and this factor tends toward the desired $\omega_{opt}$. Then if the over-relaxation ever continued for more than 50,000 over-relaxation iterations without converging, the $\rho_{Jacobi}$ was reduced by a small factor of .05 causing a reduction in the over-relaxation parameter. The relaxation was then begun anew. If further reductions in $\rho_{Jacobi}$ were necessary, they were performed and the potential recalculated until convergence was obtained. Realizing that only certain moments in the simulation required this reduced over-relaxation factor and in order to optimize the speed of the code, after 500 converged iterations passed in which $\rho_{Jacobi}$ had not been reduced, it was restored to its original level, once again yielding an $\omega$ of 1.941.

With these optimizations it was expected that the rate of convergence of the SOR algorithm would be increased significantly. Figure 5-7 shows the results of several trials conducted to illustrate this point. In the top of the figure, Figures 5-7(a) and 5-7(b) depict the residue of the convergence versus the over-relaxation iteration number. This data was taken at a particular representative simulation iteration. The bottom of the figure, Figures 5-7(c) and 5-7(d), show the convergence rate of the SOR with Chebyshev acceleration added. Note that the vertical axes of these plots are not the same. In fact, the maximum residue on the top plots is on the order of $10^{-2}$ while the maximum residue when Chebyshev acceleration was added was approximately $10^{-5}$. Without the acceleration procedure, approximately 1600 SOR iterations were required to reach the desired level of

convergence while only about 1300 iterations were needed when the acceleration was added. This translates to a significant savings in computational time.

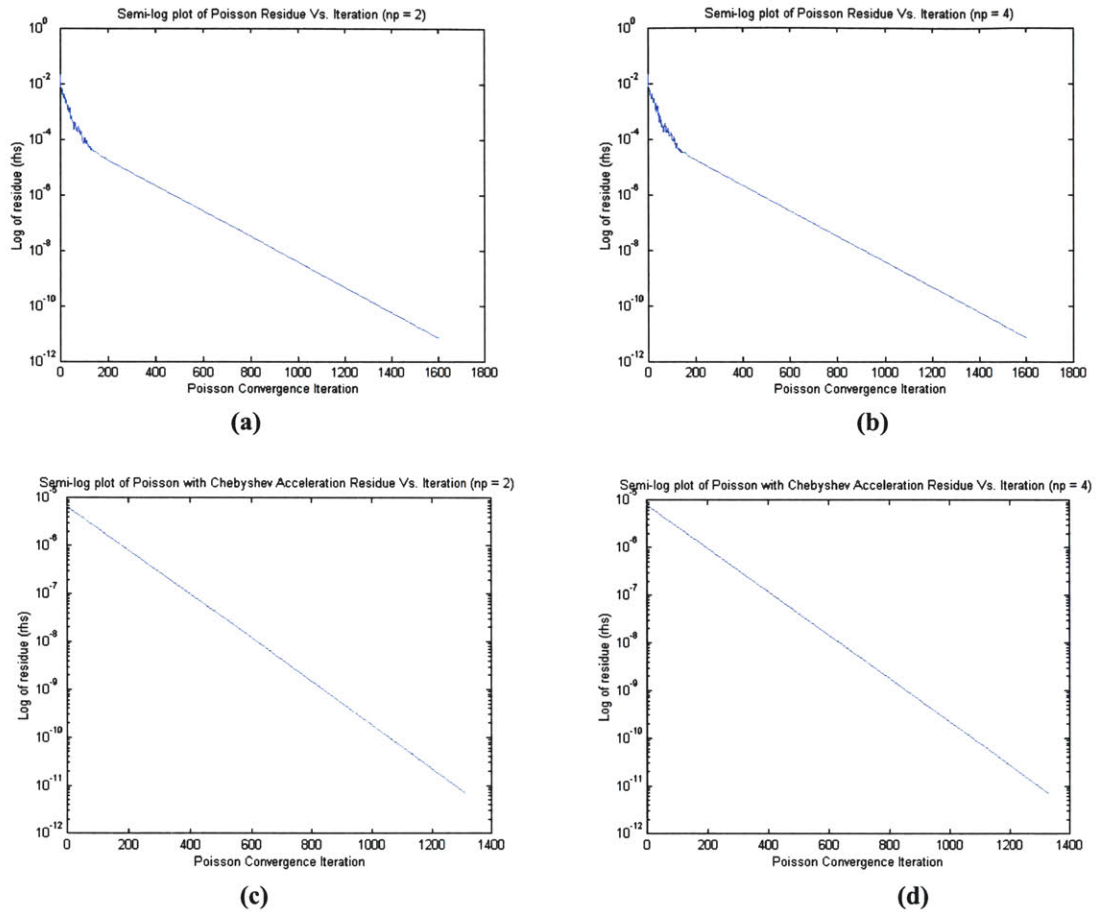



**Figure 5-7: The convergence rate of SOR without Chebyshev acceleration is shown for (a) 2 processors and (b) 4 processors. The convergence rate for SOR with Chebyshev acceleration added is then shown for (c) 2 processors and (d) 4 procesors.**

# Chapter 6

# Investigations Using the Parallel Code

## 6.1 Implementation of a Variable Hall Parameter

It has long been theorized that the anomalous electron transport coefficient does not remain constant throughout the entire thruster chamber but most likely changes with axial position. Until now, the MIT PIC simulation has, however, approximated this changing anomalous coefficient as a single constant value. It was believed that this approximation was minor and did not significantly affect simulation results. When the recent results displayed in [1] seemed to contradict this assumption, it was decided that an axially-varying Bohm coefficient should be added to the MIT PIC simulation.

### 6.1.1 Structure of Varying Hall Parameter

The authors of [1] were fortunate in that experimental measurements of the Hall parameter had already been conducted for the thruster that they chose to model. No such data could be located for the P5 thruster upon which the current work was initially focused. It was necessary then to estimate the form of the Hall parameter as it varied through the thruster chamber.

Figure 6-1(a), courtesy of [1], shows experimental measurements of the Hall parameter for the Stanford Hall thruster at three different voltages. It can be observed that as voltage is increased, the parameter increases and becomes more peaked. This implies that for our P5 thruster operating at 500 Volts, the peak should be relatively thin and the peak value should be somewhat greater than those shown in the figure. Also, noting that the exit of the acceleration channel for the Stanford Hall thruster occurs at approximately .08 meters, it is seen that the parameter peaks somewhere before the chamber exit and slightly after three-quarters of the distance to the exit. With these facts in mind, the structure shown in Figure 6-1(b) was devised as a simple likely estimate of the Hall parameter variation in the chamber of the P5, remembering that the exit of the P5's acceleration channel occurs at .04 meters. Considering the crudeness of these

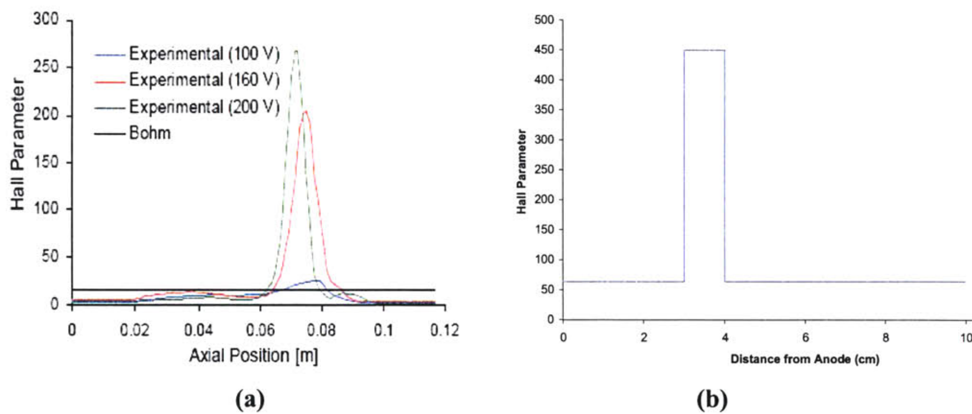



**Figure 6-1: (a) The experimentally-measured Hall parameter variation for the Stanford Hall Thruster at three different voltages and (b) the estimated Hall parameter variation for the P5 thruster at 500 Volts.**

estimates, it was believed that a simple step function would be sufficiently complex to model the variation. Given experimental data for the P5's Hall parameter, perhaps a more complicated profile could later be used.

### 6.1.2 Results Comparing Variable and Constant Hall Parameter

Surprisingly, using even the simple, highly-speculative variation in Hall parameter discussed above, exceptionally encouraging results were obtained. In fact, this addition to the code corrected discrepancies between P5 thruster simulated results and experimental results that had troubled the MIT team for years.

Comparisons were made between experimental P5 thruster data taken at the University of Michigan [17], the MIT PIC code with a constant Hall parameter of 400 (judged by Sullivan [25] to be optimum), and the MIT PIC simulation with the variable Hall parameter as discussed above. All trials were conducted using or simulating the P5 thruster running at 300 Volts with an anode mass flow rate of 10.7 mg/s. The results are shown in the table and figures that follow.

**Table 6-1: Performance characteristics for the P5 thruster**

| | Current (A) | $I_{SP}$ (s) | Thrust (mN) |
|---|---|---|---|
| **Experiment** | 10.0 | 1670 | 180 |
| **Variable Hall Parameter Simulation** | 10.2 | 1730 | 180 |
| **Constant Hall Parameter Simulation** | 7.5 | 1600 | 180 |

The general performance characteristics measured are shown in Table 6-1. The simulation with constant Hall parameter quite severely underpredicted the average

current while the variable parameter trial obtained results much nearer the mark. In terms of $I_{SP}$ and thrust, however, the simulations both obtained results of similar accuracy with respect to the experimental data.

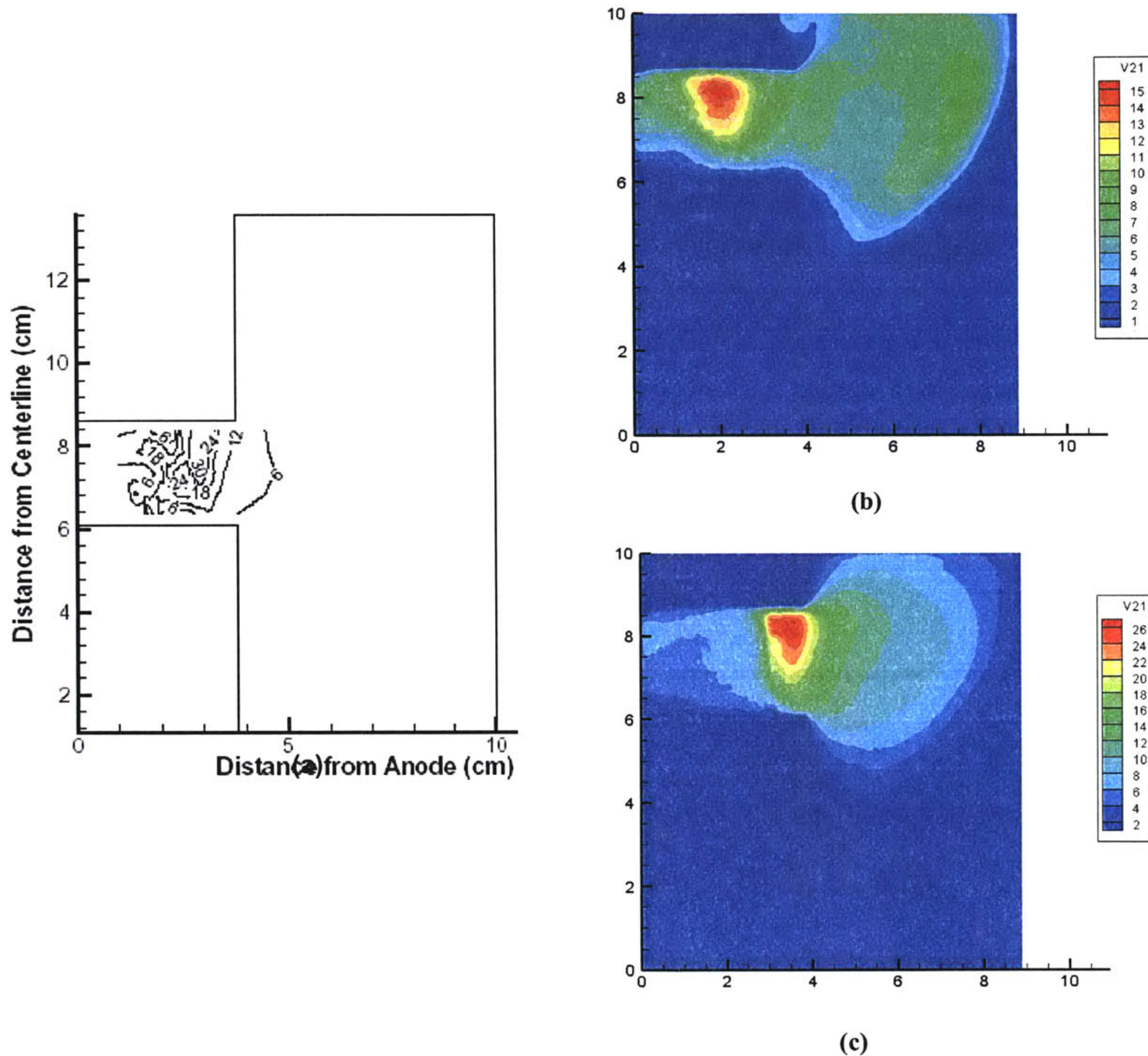



**Figure 6-2: The time-averaged electron energy (eV) for (a) experimental results from the University of Michigan, (b) the MIT PIC simulation with constant Hall parameter, and (c) the MIT PIC simulation with a variable Hall parameter.**

Figure 6-2 above compares the time-averaged electron temperature in each of the three cases. The most important feature of note is the scale on the constant Hall parameter MIT PIC simulation in Figure 6-2(b). The maximum electron temperature is

given as approximately 15 eV. This is one-half the maximum shown in the University of Michigan results presented in Figure 6-2(a). The addition of a variable Hall parameter boosts this maximum temperature to a much more accurate 26 eV while simultaneously moving the high temperature ionization region much closer to the chamber exit. This can be seen in Figure 6-2(c). Previous MIT simulations with a constant Hall parameter characteristically smeared this ionization region and placed it much closer to the anode than experimental observations indicated. With a variable Hall parameter the temperature peak is much sharper and located much closer to the expected position, the top of the acceleration chamber near the exit.

Figure 6-3 below depicts the time-averaged electric potential for each of the three cases. The results of the MIT PIC simulation with constant Hall parameter, shown in Figure 6-3(b), exhibit a relatively smooth potential gradient. This problem has been indicative of many Hall thruster simulations in the past. Indeed even the P5 thruster simulation created by [17] reported a similar inaccuracy when they compared their simulation to their experimental results. The potential tends to drop quickly near the anode and then continues to fall smoothly throughout the acceleration chamber. However, experimental observations in Figure 6-3(a) show that the potential actually remains above 250 Volts until approximately 2.5 cm down the channel and then falls rapidly off until the chamber exit is reached. The MIT PIC simulation with variable Hall parameter shows behavior more comparable to experiment, with a high voltage until approximately 2.5 cm, and then a very high gradient until the thruster exit. Clearly the addition of a variable parameter is a breakthrough in the simulating accuracy of the MIT PIC code. Of course, the profile we have selected for the Hall parameter remains semi-

empirical. Real predictive capability will likely require a self-consistent three-dimensional calculation of cross-field transport.

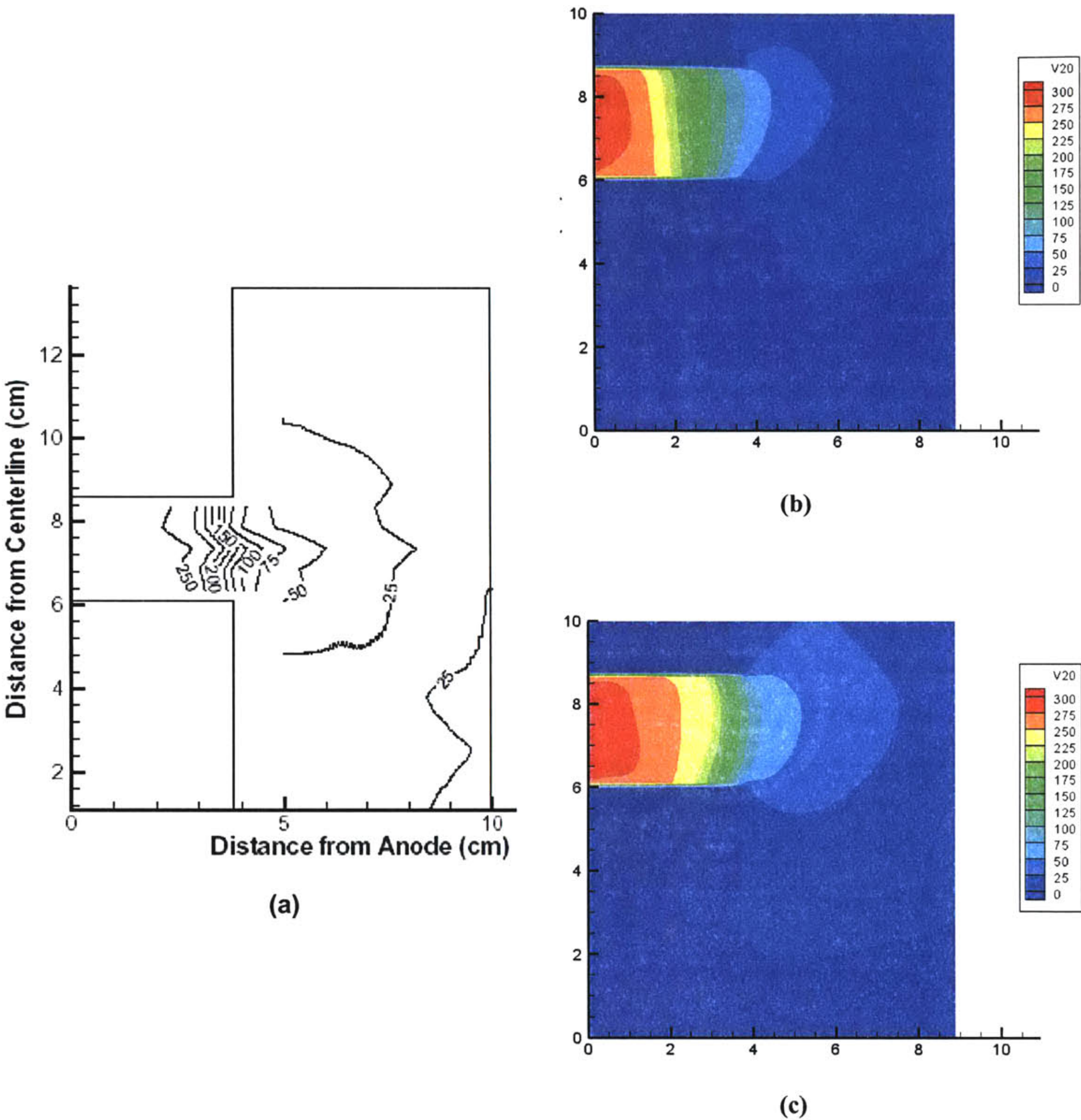



**Figure 6-3: The time-averaged electric potential for (a) the experimental University of Michigan data, (b) the MIT PIC simulation with constant Hall parameter, and (c) the MIT PIC simulation with variable Hall parameter.**

### 6.1.3 Comparing Different Hall Parameter Structures

We were well aware that our choice of a Hall parameter structure, while based loosely on the evidence from the Stanford Hall thruster shown previously in Figure 6-1(a), was more or less ad hoc. Therefore we endeavored to explore several other possible structures and to examine the impact various alterations in the Hall parameter had on thruster operation. Three such trials were attempted and are discussed below.

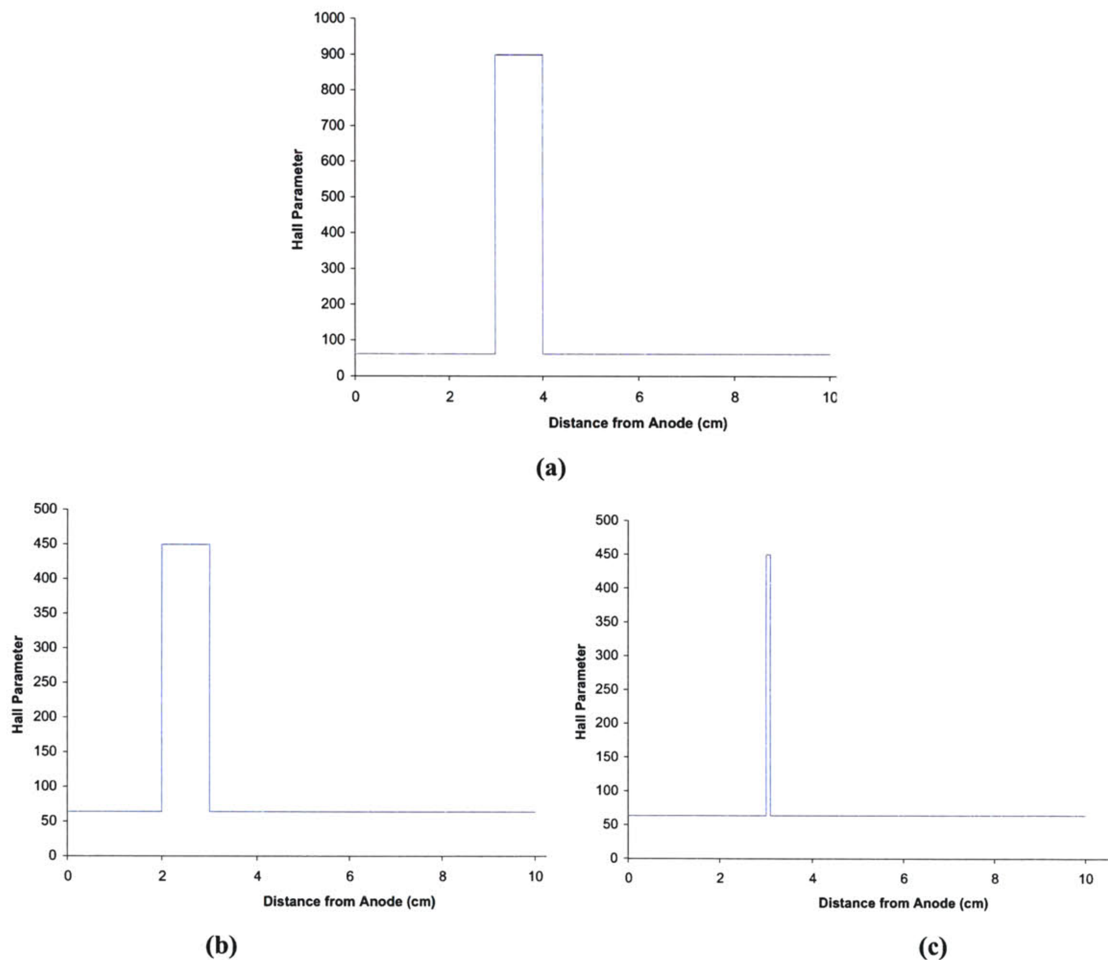



**Figure 6-4: Three Hall parameter profiles studied. Each is similar to that used in section 6.1.2, except that (a) has a higher peak, (b) has been shifted left, and (c) has a narrow peak.**

Figure 6-4 above depicts the Hall parameter shape for each of the three trials. All three structures are similar to that used in Section 6.1.2 except for a small change in each case. The trial that used the parameter shown in Figure 6-4(a) investigated the effect of a

peak which is higher than that used in Section 6.1.2. The second trial used a structure like that shown in Figure 6-4(b) which merely shifted the peak location one centimeter in the negative-z direction. Finally, the third trial examined the effect of significantly narrowing the parameter peak width via a profile like that shown in Figure 6-4(c).

From the results of these three trials depicted in Figure 6-5 below, several observations can be made. First, when the peak parameter was increased from 450 to 900, thus decreasing the level of anomalous diffusion in the ionization region, the peak in temperature became sharper and more defined. This is clearly seen in Figure 6-5(a). The temperature of the electron flow in experiment falls quickly to a value of 6 eV or less once outside the chamber. This trial exhibited a similar trend. Secondly, when the Hall parameter peak was shifted slightly left, the ionization region followed. This is clearly seen in Figure 6-5(c). In addition, the steep gradient in potential is usually exhibited in experiment at the acceleration channel exit just after the ionization region. However, this gradient too was shifted left when the Hall parameter peak was shifted left, as can be seen in Figure 6-5(d). Finally, the third trial seemed to indicate that the width of the Hall parameter peak must be greater than a certain width in order to prove effective. With a narrow peak, the desired ionization region did not form and the peak temperature remained far too low (Figure 6-5(e)). Of course, Figure 6-5(f) shows that this also led to the potential decreasing at a fairly constant rate instead of the rapid descent experiment attributes to the ionization region. Overall, from these trials it was learned that for the P5 thruster, a very high, relatively broad peak in Hall parameter was required to achieve the optimum agreement with experiment.

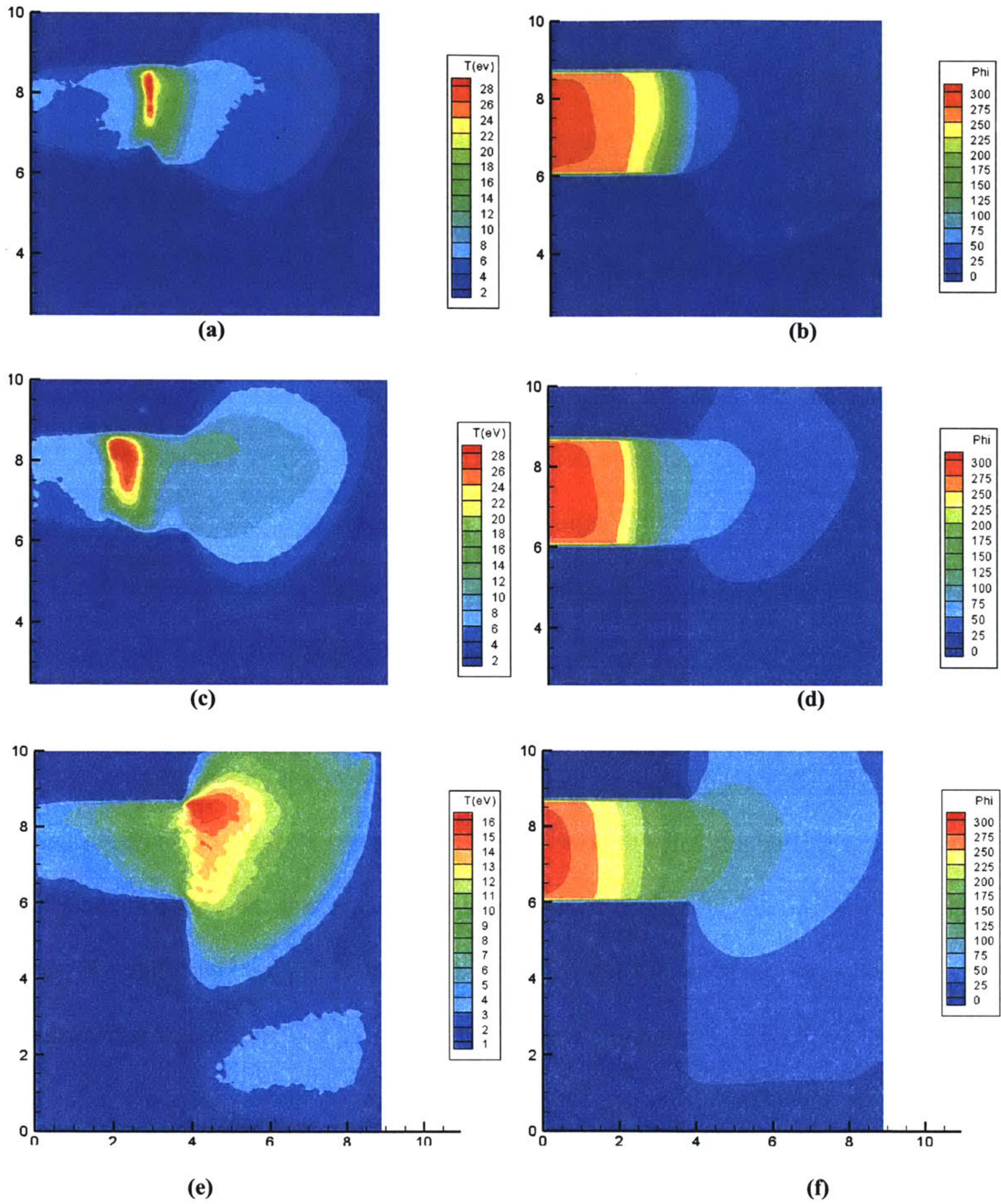



**Figure 6-5: Electron temperature and potential for the three different Hall parameter profiles: (a)(b) the higher peak, (c)(d) the left-shifted peak, (e)(f) and the narrow peak.**

## 6.2 Modeling of a High Power Central Cathode Thruster

With propulsion systems for interplanetary missions in demand, high power Hall thrusters and ion engines have recently become the topic of budding research. Many companies and research groups have begun investigating the prospect of creating thrusters within the 20 to 40 kW power range. We acquired information on one such high power Hall thruster which is still in its redesign and optimization phase. As a proprietary favor, we will be very general in our descriptions and analysis of said thruster. This relatively large thruster is also unique due to its centrally-placed cathode. Modeling such a thruster offered an interesting test of the new parallel code's utility because of the large required simulation region. It also offered an exciting opportunity to model the actual dynamics of the thruster with the electron-emitting cathode included in the simulation domain rather than merely modeled as a quasi-neutral boundary condition.

### 6.2.1 The Central Cathode Thruster and Simulation Grid

With the goals mentioned above in mind, we set about adapting the MIT parallel PIC code to model a high power, central-cathode thruster. Having acquired thruster geometry diagrams, our first task was to create a suitable simulation grid. Originally, a grid like that shown in Figure 6-6(a) was developed. The spacing was made very fine in the acceleration channel interior and allowed to grow coarser in the downstream region.

The indentation for the central cathode near the thruster's centerline, however, proved to be a difficult region to properly grid using a quadrilateral mesh. There are simply too many sides to the interior of this thruster for a proper Euclidean mesh. The problem might have been alleviated by the creation of interior object meshes, but unfortunately the MIT PIC code does not yet support sloped internal object edges. In

addition, it was put forth that the densities very near the cathode injection site would be extremely high, requiring an extremely fine mesh to properly resolve. In order to avoid these difficulties, the final gridding settled upon was that shown in Figure 6-6(b). The injection of electrons and neutrals flowing from the cathode would be handled as an already uniformly diffused plasma all along the straight boundary adjacent to the cathode.

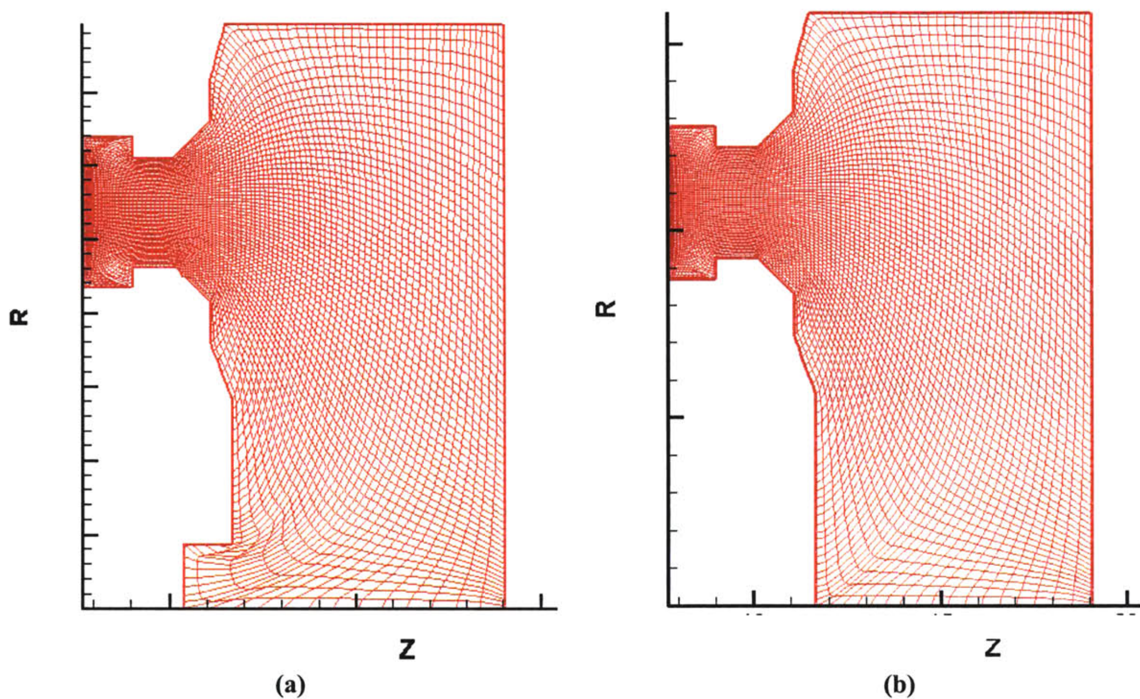



**Figure 6-6: Gridding of the high power central-cathode thruster. (a) Originally an indentation was left near the centerline for the cathode, but (b) the final grid does not explicitly model the near-cathode region.**

### 6.2.2 High Power Thruster Magnetic Field Structure

We were also able to obtain the magnetic field structure of the new high power central-cathode thruster. Some streamlines of the magnetic field are plotted below in Figure 6-7. Again, exact field strength information and thruster dimensions have been omitted as a professional courtesy. An important feature of note is the direction of the magnetic field lines near the centerline of the thruster, those which emanate from the central cathode. Unfortunately, these lines have a strong tendency to hold onto the emitted electrons and direct them straight out the right hand side of our simulation region. In reality, these electrons would follow largely along the magnetic field lines, drifting slightly inward due to the potential gradient toward the anode, and strike the thruster wall somewhere above our simulation region. The corresponding thruster surface in this unsimulated region would soon develop an electron-repelling sheath which would bounce

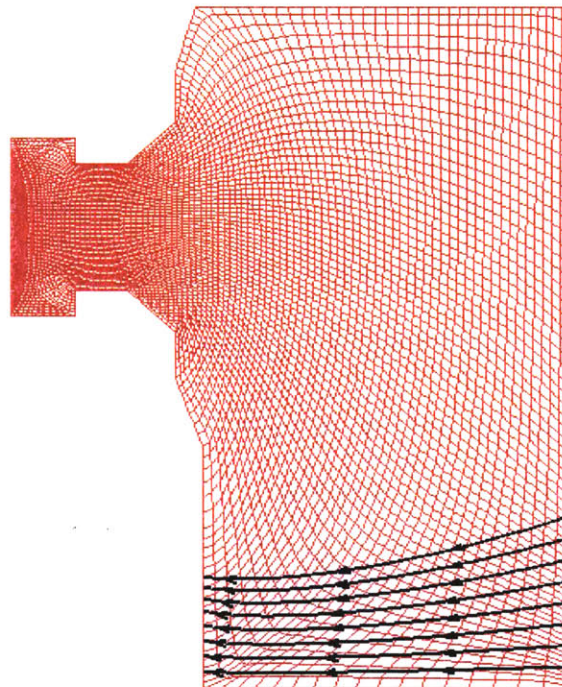


**Figure 6-7: The magnetic field structure of the high power central-cathode thruster.**

the electrons back along the magnetic field lines, again with some anode-directed diffusion. They would then re-emerge on the right hand side of our simulation region, most likely at a point slightly higher than they exited.

It was for this reason that we decided to reinject any electron that reached the right hand side boundary in the region directly opposite the central cathode, moving it upward a small random amount in the radial direction and reversing its velocity. In addition, its energy was scaled to account for the potential change it would have undergone had it exited the region at a particular potential and re-entered at a different, slightly higher potential.

This is certainly a slightly ad hoc solution, but such a work-around technique would be necessary almost no matter how large the simulated geometry was allowed to be. The magnetic field lines near the centerline are nearly parallel to the central axis; at least some electrons would undoubtedly be lost to the right hand boundary. This issue generally has not been addressed because in a normal, non-central cathode arrangement, the plasma density near the centerline is typically very low and the corrections would be negligible. Thus, while our choice of an axisymmetric central cathode thruster made explicit modeling of the cathode-thruster interactions possible with a two dimensional simulation, it also caused its own set of modeling difficulties. These difficulties are noticeable in our modeling results which follow.

### 6.2.3 Results of High Power Thruster Modeling

After a great deal of experimentation and iteration, the right hand side reinjection boundary condition was finally settled upon as achieving the most believable results for the high power thruster operation. Unfortunately, this thruster is still in its early stages of development, and as such neither detailed performance specifications nor experimental density, temperature, or potential measurements with which we might compare our computations are currently available. Therefore, the results of the modeling will be presented with some minimal qualitative discussion as to their probable correctness and implications for the thruster's operation, but detailed numerical comparisons between experiment and simulation will have to wait for experimental results to be obtained.

Taking into account the results of the work featured in Section 6.1 showing the importance of a variable Hall Parameter, a Hall Parameter structure with a peak of 450 in the acceleration chamber region was used to obtain all of the following results. Outside of this region, the parameter was set to a constant low value of 64. More experimentation into the effects of this choice may be conducted by future investigators, but for our immediate purposes, a rough estimate such as this was sufficient.

Figure 6-8 depicts a time-averaged image of the high power thruster's electric potential. The anode is biased to 500 Volts as is usual for this thruster. The thruster surface region representing the central cathode exit has been held to a constant -12 Volts, a value experimentally observed to be an average operating value for this thruster. The potential appears reasonable for the most part. In the early part of the acceleration chamber region, the gradient is extremely steep, indicating the presence of an ionization zone there. The potential then quickly falls off to a nominal value of about 25 Volts

outside of the chamber. There is a slight potential gradient directing electrons away from the central cathode and upward toward the ionization zone, a result that also seems to indicate that the thruster is performing as planned.

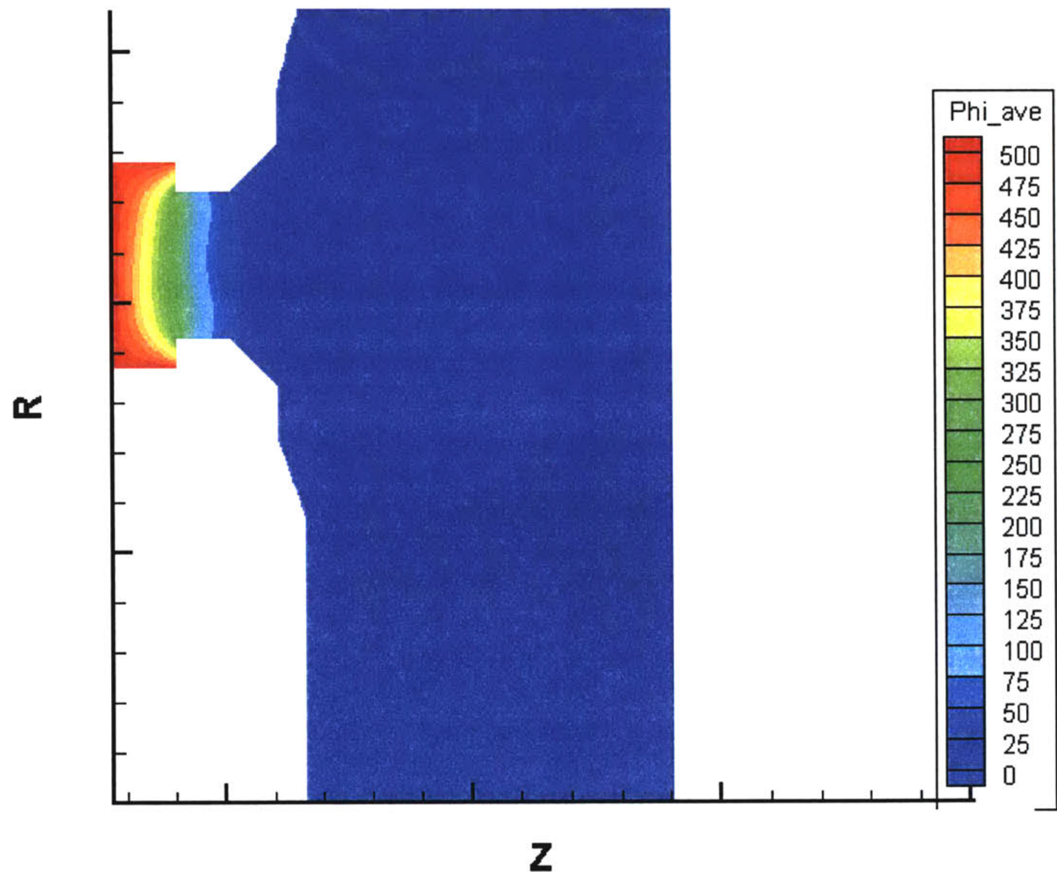



**Figure 6-8: Time-averaged electric potential for the high power thruster**

The next result of interest is the density of neutrals in the simulation region, depicted by Figure 6-9. In the center of the acceleration chamber region, a very noticeable dearth of neutrals is observed, with the minimum density dropping below $2*10^{12}$ $cm^{-3}$. This hole in the density profile is consistent with the formation of a typical ionization region, further indicating proper operation of the thruster. It is this lack of

neutral ionization targets in the acceleration chamber that leads to the breathing mode oscillations of Hall thrusters which have seen so much attention and study [3].

Another item of interest in Figure 6-9 is the lack of a neutral plume exiting the central cathode. The density is on the order of $10^{13}$ $cm^{-3}$ just outside the cathode, but quickly falls off by at least an order of magnitude. This is probably simply indicative of the gas expanding into a relative vacuum and does not necessarily imply a region of high ionization as it did in the thruster channel.

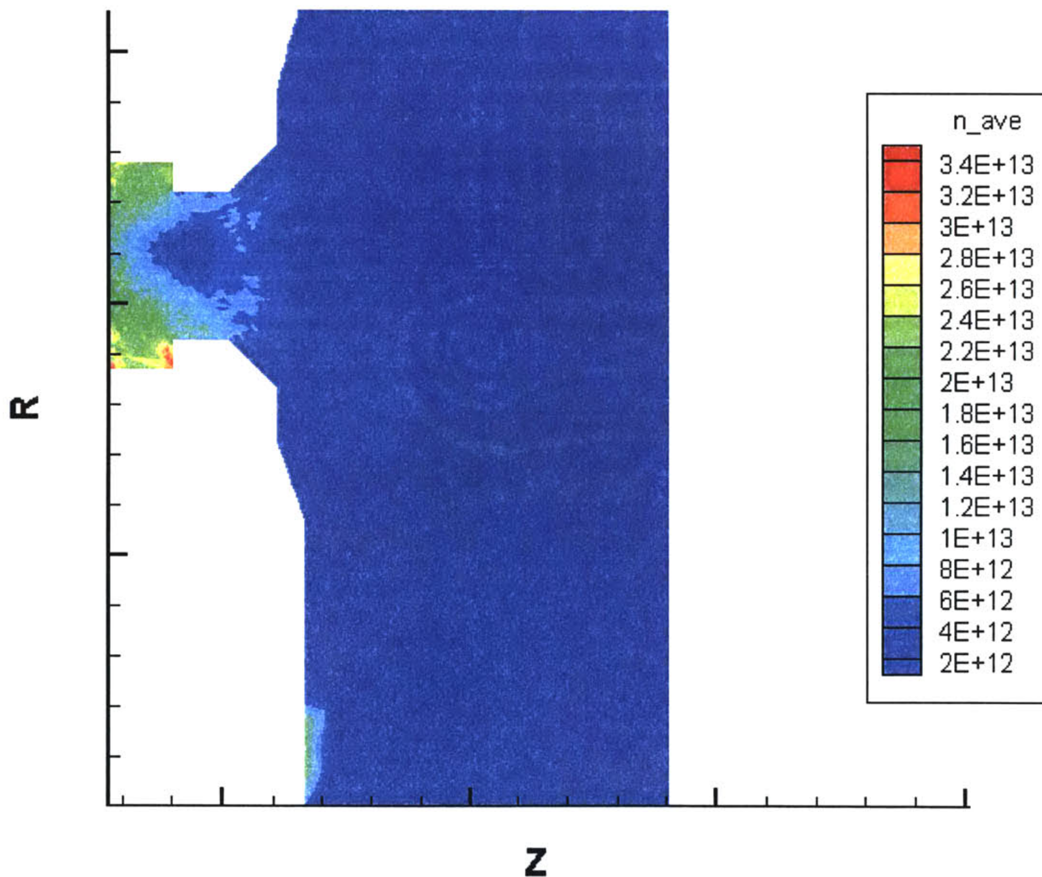



**Figure 6-9: Time-averaged neutral density ($cm^{-3}$) for the high power thruster**

The next figure, Figure 6-10, shows the ion temperature in the thruster, averaged over time. Clearly visible is the high temperature ion jet issuing from the acceleration

channel of the thruster. The jet appears to be quite focused with peak temperatures on the order of 43 eV. It is as yet unknown whether the ions actually reach such a high temperature in the high power thruster, and until further experimental results are obtained it is difficult to judge the accuracy of these values. Temperatures on the right boundary of the simulation region between the ion jet and the central cathode appear to be larger than would be physically expected. This is most likely due to our ad hoc electron reinjection boundary condition. Cleaner results could be expected with a larger simulation domain or a more accurate representation of the electron physics outside the simulation region.

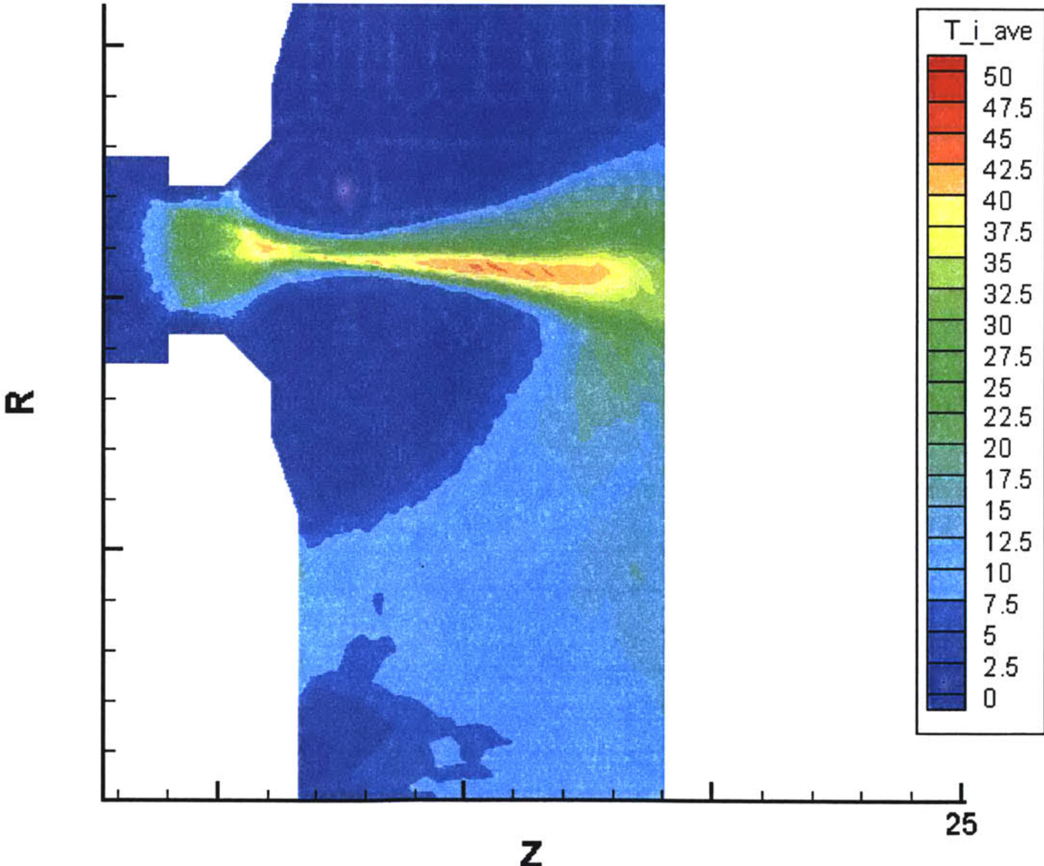



**Figure 6-10: Time-averaged ion temperature (eV) for the high power thruster**

Figure 6-11 below shows the time-averaged electron temperature for the thruster. Peak temperatures of 65 eV are achieved locally within the acceleration chamber of the thruster. Such a peak is indicative of a strong ionization region. After this peak, the temperature falls quickly off to a nominal value and remains low until the right side of the simulation region is reached. Unfortunately, again due to our abbreviated simulation domain and the estimated nature of our reinjected electron condition, the temperature rises along an arc near the right hand side of the simulation. This temperature rise is believed to be purely numerical in nature and does not represent any actual physical process. It is expected, again, that larger simulations, or better yet a three-dimensional simulation with off-axis cathode, would yield cleaner modeling results.

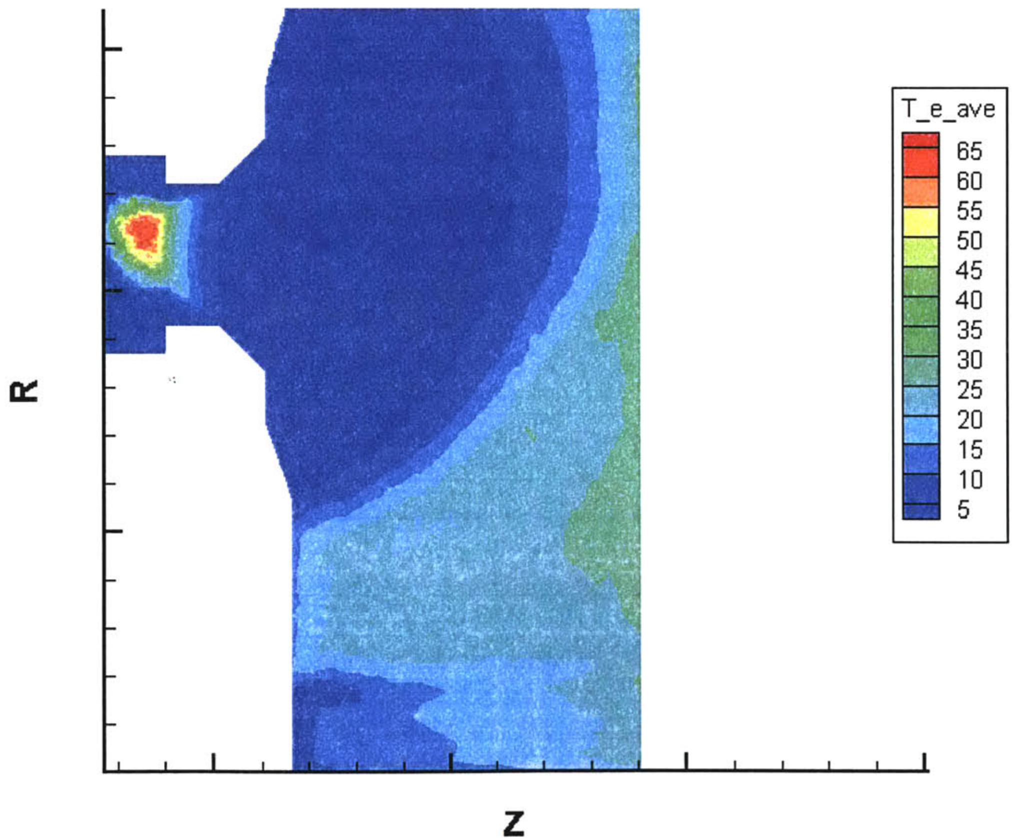



**Figure 6-11: Time-averaged electron temperature (eV) for the high power thruster**

### 6.2.4 Discussion of the High Power Thruster and Results

There were two driving reasons for selecting this particular thruster for modeling. First of all, it was a relatively high power thruster. This meant that the scale of simulation would be larger. Fully kinetic simulations of this thruster without the parallel code might require as much as 35 days to complete. The parallel code was able to reduce this number to approximately 10 days while running on just four processors. Such a demonstration of the effectiveness of parallelization helped to justify the work conducted in the previous chapters of this thesis.

The second reason for simulating the high power thruster was its axial cathode arrangement. For the first time, we would be able to simulate the cathode-thruster interactions without violating our two-dimensional axisymmetric assumption. Unfortunately, we discovered that since the magnetic field near the centerline of the thruster quickly directs electrons outside of our simulation region, the axial cathode arrangement holds its own pitfalls that complicate simulation.

However, despite the large scale of the problem and despite the necessity of some ad hoc boundary condition assumptions, the MIT full-PIC code was still able to create a reasonable picture of the microscopic workings of this high power thruster. The electric potential calculated was consistent with that seen in similar smaller thrusters [17] while the peak electron temperatures in the acceleration chamber reached reasonable levels considering the power level of this thruster. Future work with this thruster should involve obtaining detailed experimental data with which to compare our simulation results. More investigation into the boundary conditions required when an axial cathode

is modeled would also be beneficial. Finally, the effects of varying the Hall parameter in specific ways for this thruster could be further explored and understood.

# Chapter 7

# Conclusions and Future Work

## 7.1 Summary of Work

The full particle-in-cell simulation built in [26] and maintained by MIT's Space Propulsion Laboratory was successfully adapted to run on parallel processors, thus drastically reducing the time required for numerical investigations to be accomplished. It was shown that this parallelization did not reduce the accuracy of the simulation in any way. While a linear time speed up was not observed in the case of the relatively small P5 thruster, experiments were conducted showing a linear speed up in computational efficiency on problem sizes approximately 10 times as large as the P5 thruster. This work thus paves the way for future efforts at high power three-dimensional Hall thruster full-scale modeling.

While conducting this parallelization work, it was discovered that former implementations of MIT's PIC code may not have been properly converging to the correct electric potential at every time step within a simulation. Fortunately, this issue created only insignificant discrepancies in the final data and was easily corrected by the

proper choice of over-relaxation coefficient and the addition of the Chebyshev acceleration technique to the potential solver.

Next, the fully-parallelized code was used to conduct some investigations which helped to highlight its capabilities and usefulness. The first of these experiments delved into the possible effects the Hall parameter spatial profile, and thus electron mobility, might have upon thruster operation. It was shown that experimental results were most closely matched not by a constant Hall parameter as had been previously assumed in the MIT PIC simulation, but by a peaked structure which reduced electron mobility in the region of the acceleration chamber. These results helped to correct some mismatches between experimental results and computational results mentioned by [5] and [25].

Finally, a high power central-cathode thruster currently in the initial stages of development was modeled. This high power thruster tested the efficiency of the parallelization which, by using only four processors, reduced an almost unmanageable 5 week computation to one requiring just over a week to complete. This thruster also gave MIT's PIC code its first opportunity to explicitly model cathode-thruster interactions by including the cathode in the simulation region. Results obtained for this thruster appeared qualitatively consistent with available experimental data. However, a more detailed analysis was not readily possible owing to the fact that the high power thruster is still in development and does not yet have a body of experimental results available for detailed comparison.

## 7.2 Possible Future Work

Some possible suggestions for further inquiry:

- Addition to the code of a model which calculates self-consistently the Hall parameter at each grid cell within the simulation. Work has been suggested which might relate the axial gradient of the Hall current and electric field shear to the electron mobility. (See, for example, [6]).

- Extension of the parallel PIC technique to a fully three-dimensional simulation is imperative. Certain phenomena, such as the variation in Hall parameter, may be in actuality three-dimensional effects, and it is important that these effects be included in future modeling efforts as their impact has been shown to be significant. Parallelization makes a three dimensional effort feasible, but more sophisticated computational algorithms or meshing geometries may be required to make the effort efficient.

- The use of Successive Over-Relaxation and an irregular mesh could be re-examined. Perhaps a fast Fourier Transform on a regular mesh might be considered in the future to significantly speed up computation. Adaptive meshing techniques could also be investigated.

- Further investigation of the high power thruster. Comparisons with experimental results should be made once those results are available. In addition, the ad hoc boundary conditions used to reinject electrons exiting along the magnetic field could be further researched and understood. Perhaps an enlarged simulation region could also alleviate much of this problem.

- Simulation of even larger, higher power Hall thrusters that have a body of experimental work with which to compare computational results or that are being designed and require theoretical understanding and engineering predictions to optimize their performance.

# APPENDIX A

# Useful MPICH functions

**MPI_Send**: The basic send message command in MPICH. A call to this function will not return until a matching MPI_Recv has been called on the appropriate destination process. No data is altered on the sending process.

*MPI_Send(void ***Msg**, int **Length**, MPI_Datatype **Datatype**, int **Destination**, int **Tag**, MPI_Comm **Communicator**)*

***Msg**: A pointer to the start of the contiguous memory block to be transferred.*

***Length**: The number of items of type **datatype** to transfer. Therefore, an MPI_Send of 3 MPI_INTs will transfer 3*sizeof(int) bytes of data.*

***Datatype**: One of the pre-defined MPI_types which are generally identified simply by the standard C types affixed with the "MPI_" prefix. Note: sizeof(int) and sizeof(MPI_INT) are not necessarily equal.*

***Destination**: the pid of the receiving processor.*

***Tag**: must match in corresponding MPI_Send and MPI_Recv calls. The tag can be used by the programmer as a flag for logical control, for instance.*

***Communicator**: specifies which group of processes can "hear" this message. Creating smaller groups may help increase message-passing efficiency.*

Ex: `MPI_Send(&int_array[5][0], 15, MPI_INT, 1, 0, MPI_COMM_WORLD);`

Assuming that int_array is an initialized 10x10 matrix on the processor which made this call, the above statement will attempt to tell processor 1 the values of the integers stored in int_array[5][0] through int_array[6][4].

**MPI-Recv**: The basic receive message command in MPICH. A call to this function will not return until a matching MPI_Send has been called on the appropriate sending process. The data in ***Msg*** need not be initialized, but must be of the proper type or a Floating Point Exception or Message Truncation error may occur. Any previous data stored in ***Msg*** will be overwritten by the sent values.

*MPI_Recv(void* **Msg,** int **Length,** MPI_Datatype **Datatype**, int **Sender,** int **Tag**, MPI_Comm **Communicator,** MPI_Status* **Status)***

***Msg**: A pointer to the start of the contiguous memory block where incoming information will be written.*

***Length**: The number of items of type **datatype** to accept.*

*The receiving buffer must be at least as large as the send buffer, but need not be exactly the same length.*

***Datatype**: One of the pre-defined MPI_types which are generally identified simply by the standard C types affixed with the "MPI_" prefix. Note: sizeof(int) and sizeof(MPI_INT) are not necessarily equal.*

***Sender**: the pid of the sending processor.*

***Tag**: must match in corresponding MPI_Send and MPI_Recv calls. The tag can be used by the programmer as a flag for logical control, for instance.*

***Communicator**: specifies which group of processes can "hear" this message. Creating smaller groups may help increase message-passing efficiency.*

***Status**: A pointer to an MPI_Status object defined on the receiving processor. This object can be queried to discern information about the received message. This is used more for non-blocking operations of which there are none in this project.*

Ex: MPI_Recv(double_array, 20, MPI_DOUBLE, 3, 8, MPI_COMM_WORLD, &status);

Assuming that double_array is a 10x10 matrix on the processor which made this call, the above statement will wait for an MPI_Send command with tag = 8 to be executed on processor 3.

It will then place as many doubles as are sent (up to 20) into double_array[0][0] through double_array[1][9].

**MPI-Bcast**: This function is used to efficiently broadcast data from a root node to all the other processors in a group. This function will not return until every processor in the group has made the same call to *MPI_Bcast*.

*MPI_Bcast(void** ***Msg****, int* ***Length****, MPI_Datatype* ***Datatype****, int* ***Root****,*
*MPI_Comm* ***Communicator****)*

***Msg****: On the root processor, this is a pointer to the start of the contiguous memory block to be sent to the other processors. It will not be altered on the root processor. On all other processors, this is a pointer to the start of the contiguous memory block where the root processor's data will be received. After the call, the data stored at* ***Msg*** *by every processor in the group will match that stored at the root .*

***Length****: The number of items of type* ***datatype*** *to be sent and received.*

***Datatype****: One of the pre-defined MPI_types which are generally identified simply by the standard C types affixed with the "MPI_" prefix. Note: sizeof(int) and sizeof(MPI_INT) are not necessarily equal.*

***Root****: the pid of the root processor. In our case, this will always*

*be 0.*

***Communicator**: specifies which group of processes can "hear" this message. Creating smaller groups may help increase message-passing efficiency.*

Ex: `MPI_Bcast(double_array, 20, MPI_DOUBLE, 0, MPI_COMM_WORLD);`

Assuming that double_array is a 10x10 matrix defined on all processors in the group MPI_COMM_WORLD, the above statement will wait for all processors to execute the corresponding MPI_Bcast command. It will then copy the values of the doubles contained on processor 0 in double_array[0][0] through double_array[1][9]. On all other processors in the group, the copied values will then be placed into double_array[0][0] through double_array[1][9].

**MPI-Reduce**: This command allows values to be operated on across a group of processors and the result to be stored at the root node. A call to this function will not return until every processor in the group has executed the proper MPI_Reduce call. The data stored in ***Values*** will not be altered in any processor, and the data stored in ***Result*** will only be altered on the root processor.

*MPI_Reduce(void* **Values,** void* **Result,** int **Length, MPI_Datatype Datatype,** MPI_Op **Operator,** int **Root,** MPI_Comm **Communicator)***

***Values**: A pointer to the start of the contiguous memory block*

*containing the values upon which the reduction will be performed. This memory space must be initialized on all processors, but will not be altered.*

***Result:*** *A pointer to the start of the contiguous memory block into which the results of the operation will be placed. Note that* ***Result*** *and* ***Values*** *may not overlap on the root processor. This data will only be altered on the root processor.*

***Length****: The number of items of type* ***datatype*** *to reduce.*

***Datatype****: One of the pre-defined MPI_types which are generally identified simply by the standard C types affixed with the "MPI_" prefix. Note: sizeof(int) and sizeof(MPI_INT) are not necessarily equal. Some MPI_Ops may be used on only certain types of data.*

***Operator:*** *Any of a number of predefined or user-defined operations that combine the data of all processors in the group. In the case where* ***Values*** *and* ***Result*** *are arrays, the operation is conducted elementwise, once for each element in* ***Values,*** *and stored in the corresponding elements of* ***Result.***

***Root****: the pid of the root processor on which the result of the reduction will be stored.*

***Communicator****: specifies which group of processes can "hear"*

*this message. Creating smaller groups may help increase message-passing efficiency.*

Ex: MPI_Reduce(sums_of_scores, my_test_scores, 3, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

Imagine we wanted to find the class average on a series of three quizzes, and imagine that each processor in the group contains the test scores of one of the students in the class in the 3x1 integer array my_test_scores. The above command would then sum the value of my_test_scores[1] across all the processors in the group and store it on processor 0 at sum_of_scores[1]. Similarly with my_test_scores[0] and my_test_scores[2]. In this way the class total on each of the three tests could be efficiently found.

**MPI-Sendrecv**: This function allows a message to be sent and received by a processor using only a single call. It also allows the send and receive to be performed in either order, ensuring the maximum possible efficiency. It is particularly useful in our Gauss Law solver since boundary values must be passed and received by every processor at each iteration. A call to this function will not return until both a matching MPI_Send and an MPI_Recv have been called on the appropriate processors. Any previous data stored in ***MsgToRcv*** will be overwritten by the sent values, but the values in ***MsgToSend*** will be unaltered.

*MPI_Sendrecv(void** ***MsgToSend,*** *int* ***SendLength,*** *MPI_Datatype* ***SendDatatype,*** *int* ***Destination,*** *int* ***SendTag,*** *void** ***MsgToRcv,*** *int*

***RcvLength,*** *MPI_Datatype* ***RcvDatatype,*** *int* ***Sender,*** *int* ***RcvTag,***
*MPI_Comm* **Communicator,** *MPI_Status** **Status)**

***MsgToSend:*** *A pointer to the start of the contiguous memory block to be sent to another process.*

***SendLength:*** *The number of items of type* **datatype** *to transmit.*

***SendDatatype:*** *One of the pre-defined MPI_types which are generally identified simply by the standard C types affixed with the "MPI_" prefix. Note: sizeof(int) and sizeof(MPI_INT) are not necessarily equal.*

***Destination:*** *the pid of the processor to receive this message.*

***SendTag:*** *must match in corresponding MPI_Recv call on destination processor. The tag can be used by the programmer as a flag for logical control, for instance.*

***MsgToRcv:*** *A pointer to the start of the contiguous memory block to be received from another processor.*

***RcvLength:*** *The number of items of type* **datatype** *to be received.*

***RcvDatatype:*** *One of the pre-defined MPI_types which are generally identified simply by the standard C types affixed with the "MPI_" prefix. Note: sizeof(int) and sizeof(MPI_INT) are not necessarily equal.*

***Sender:*** *the pid of the processor from which to receive this message.*

***RcvTag:*** *must match in corresponding MPI_Send call on*

*sending processor. The tag can be used by the programmer as a flag for logical control, for instance.*

**Communicator**: *specifies which group of processes can "hear" this message. Creating smaller groups may help increase message-passing efficiency.*

**Status:** : *A pointer to an MPI_Status object defined on the receiving processor. This object can be queried to discern information about the received message. This is used more for non-blocking operations of which there are none in this project.*

Ex: MPI_Sendrecv(my_top, 20, MPI_DOUBLE, myid+1, 0, my_neighbors_top, 10, MPI_DOUBLE, myid-1, 0, MPI_COMM_WORLD, &status);

If processor myid wants to tell the processor above him what his top boundary values are and wants to know what the top boundary values of the processor below him are, this would be a good function to call. The statement above sends 20 elements of the array my_top to processor (myid+1) and waits to receive 10 elements from processor (myid-1) which will then be placed in my_neighbors_top. If this same operation is occurring at almost the same time on a large number of processors, this is the most efficient method available in MPICH of sending and receiving those values.

# REFERENCES

[1] Allis, Michelle K., Nicolas Gascon, Caroline Vialard-Goudou, Mark A. Cappelli, and Eduardo Fernandez. *A Comparison of 2-D Hybrid Hall Thruster Model to Experimental Measurements.* 40th AIAA Joint Propulsion Conference, Ft. Lauderdale, FL, 2004.

[2] Altas, Irfin and Murli. M. Gupta. *Iterative Methods for Fixed Point Problems on High Performance Computing Environments.* Complexity International, Voulme 3, 1996.

[3] Barral, S., Z. Peradzynski, K. Makowski, and M. Dudeck. *Model of Gas Flow Inside a Plasma Thruster.* International Congress of Theoretical and Applied Mechanics, 2004.

[4] Blateau, Vincent, Manuel Martinez-Sanchez, and Oleg Batishchev. *A Computational Study of Internal Physical Effects in a Hall Thruster.* AIAA-2002-4105. Joint Propulsion Conference, 2002

[5] Blateau, Vincent. *PIC Simulation of a Ceramic-lined Hall-effect Thruster.* S.M. Thesis, Massachusetts Institute of Technology, 2002.

[6] Burrell, K. H. *Effects of ExB velocity shear and magnetic shear on turbulence and transport in magnetic confinement devices*, Bulletin of the American Physical Society, 1996.

[7] Celik, Murat, et. al. *Hybrid-PIC Simulation of a Hall Thruster Plume on an Unstructured Grid with DSMC Collisions.* IEPC-03-134, 31st International Electric Propulsion Conference, 2003.

[8] Dias da Cunha, Rudnei. *Parallel Overrelaxation Algorithms for Systems of Linear Equations.* Transputing, Volume 1, 1991.

[9] Fife, J. M., M. Martinez-Sanchez, and J. J. Szabo. *A Numerical Study of Low-frequency Discharge Oscillations in Hall Thrusters.* 33rd AIAA Joint Propulsion Conference, Seattle, WA, 1997.

[10] Fife, J. M., Two-Dimensional Hybrid Particle-In-Cell Modeling of Hall Thrusters. Master's Thesis, Massachusetts Institute of Technology, 1995.

[11] Fox, Justin, Shannon Cheng, Oleg Batishchev, and Manuel Martinez-Sanchez. *Fully Kinetic Modeling of a Hall-Effect Thruster with Central Cathode.* The American Physical Society's 46th Annual Meeting of the Division of Plasma Physics, 2004.

[12] Gulczinski, J. S. *Examination of the structure and evolution of ion energy properties of a 5kW class laboratory Hall effect thruster at a various operational conditions.* PhD thesis, University of Michigan, 1999.

[13] Hirakawa, M. *Electron Transport Mechanism in a Hall Thruster.* IEPC-97-021, 25th International Electric Propulsion Conference, 1997.

[14] Hirakawa, M. and Y. Arakawa. *Numerical Simulation of Plasma Particle Behavior in a Hall Thruster.* AIAA-96-3195, 32nd AIAA/ASME/SAE/ASEE Joint Propulsion Conference, Lake Buena Vista, FL, 1996.

[15] Hirakawa, M. and Y. Arakawa. *Particle Simulation of Plasma Phenomena in Hall Thrusters.* IEPC-95-164, 24th International Electric Propulsion Conference, Moscow, Russia, 1995.

[16] Khayms, V. *Advanced Propulsion for Microsatellites.* Ph.D. Thesis, Massachusetts Institute of Technology, 2000.

[17] Koo, Justin W., Iain D. Boyd, James M. Haas, and Alec D. Gallimore. *Computation of the Interior and Near-Field Flow of a 2-kW Class Hall Thruster.* AIAA-2001-3321, 37th AIAA Joint Propulsion Conference, 2001.

[18] Kuo, Jay and Tony F. Chan. *Two-color Fourier Analysis of Iterative Algorithms for Elliptic Problems with Red/Black Ordering.* USC-SIPI Report #139, 1989.

[19] Lentz, C. A. *Transient One Dimensional Numerical Simulation of Hall Thrusters.* Master's Thesis, Massachusetts Institute of Technology, 1993.

[20] Noguchi, R., M. Martinez-Sanchez, and E. Ahedo. *Linear 1-D Analysis of Oscillations in Hall Thrusters.* IEPC-99-105, 26th International Electric Propulsion Conference, 1999.

[21] Press, et. al. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1988.

[22] Roy, Subrata and Birendra Pandey. *Development of a Finite Element Based Hall Thruster Model for Sputter Yield Prediction.* IEPC-01-49. 29th International Electric Propulsion Conference, 2001.

[23] Sullivan, Kay, J. Fox, O. Batishchev, and M. Martinez-Sanchez. *Kinetic Study of Wall Effects in SPT Hall Thrusters.* 40th AIAA Joint Propulsion Conference, 2004.

[24] Sullivan, Kay, Manuel Martinez-Sanchez, Oleg Batishchev, and James Szabo. *Progress on Particle Simulation of Ceramic-Wall Hall Thrusters.* 28th International Electric Propulsion Conference, 2003.

[25] Sullivan, Kay Ueda. *PIC Simulation of SPT Hall Thrusters: High Power Operation and Wall Effects*. S.M. Thesis, Massachusetts Institute of Technology, 2004.

[26] Szabo, James J. *Fully Kinetic Numerical Modeling of a Plasma Thruster.* Ph.D. Thesis, Massachusetts Institute of Technology, 2001.

[27] Szabo, J. J., M. Martinez-Sanchez, and J. Monheiser. *Application of 2-D Hybrid PIC code to Alternative Hall Thruster Geometries.* AIAA-98-3795, 34th AIAA Joint Propulsion Conference, 1998.

[28] Szabo, J. M. Martinez-Sanchez, and O. Batishchev. *Fully Kinetic Hall Thruster Modeling.* 26th International Electric Propulsion Conference, 2001.

[29] Van der Pas, Ruud. *The PVM Implementation of a Generalized Red Black Algorithm.* Supercomputer, 1993.

[30] http://www.beowulf.org/beowulf/history.html

[31] http://ei.cs.vt.edu/~history/Parallel.html

[32] **http://www-unix.mcs.anl.gov/mpi/**

[33] **http://computer.howstuffworks.com/question697.htm**