Electric Charge Trajectory Simulation: A High Performance Computing Approach

Jose R. Medina Delgado
Computer Engineering
Yahya Masalmah, Ph.D.
Electrical & Computer Engineering and Computer Science Department
Polytechnic University of Puerto Rico

Abstract *— A computational model for representing an Electric Charge Trajectory within a magnetic field can be used to establish the basis for further development and research on the plasma physics domain. Regardless the mathematical models used for such simulations, constraints are always present. Particularly in terms of accuracy of the output, time for processing, and amount of information. This paper presents a comparison between three processing models on a Windows HPC environment using: MPI, Microsoft Task Parallel Library, and simple programming. By using concepts of object oriented programming with C#, high performance computing with parallel processing, and system integration with ParaView or MATLAB for scientific visualization, an entire solution is provided to help on understanding the basis for plasma physics by means of a high capacity computer simulation.*

Key Terms *— Object Oriented Software Design, Parallel Processing, ParaView, Plasma Physics.*

INTRODUCTION

In a software simulation, an attempt to represent a real life event is achieved by means of mathematical models. By using computers to perform these calculations, a readable output is obtained. As a result, anything that could be explained mathematically could be simulated. The importance of simulation relays in the fact that it provides a way to demonstrate the result of an event without the need of real life intervention, that otherwise would be expensive in terms of time, materials, or damages.

Moreover, computer simulations usually provide the more effective and cheaper alternative to experimental measurements [1]. A simulation could be as simple as to represent a one direction straight motion of a ball, or as complex as to represent the properties of materials under shock at a high velocity impact. In any case there are common processing constraints associated with a simulation: accuracy of the output, time for processing, and amount of information. To work with these processing constraints, a hardware platform capable to run as fast as possible is needed. Also a high amount of storage to provide the output might be needed. However, in addition to the hardware side, other challenges arise during this process.

One of these challenges that software engineers face, is to understand the domain of the problem [2]. Having certain knowledge on the domain of the problem may result in more accurate software, with less re-decoding effort or a faster delivery of the solution. This is critical, especially during the design of software for simulation. When developing software for simulation, the mathematical models that represent them have to be understood. These mathematical models are the domain requirements for the system. A major problem with domain requirements is that they are written in the language of the application domain, mathematical formulas in our case, and it is often difficult for software engineers to understand them [2]**.**

In addition to understanding the domain of the problem, determining if the system could be implemented is another challenge. As part of the software engineering process, a feasibility study to identify the technology that could be used, and the integration of existent components of hardware and software to speed up the delivery of the system was a first step on all this work. However, in the case of the simulation, some aspects of the implementation could not be determined until the domain of the problem is understood. Typically software engineers may oversee this part of the lifecycle process, ending up in a restart of the entire lifecycle process due to hardware constraints. Even if the programming code of the system may be written as 100% portable code, for simulation software, the hardware part is also very important. The performance of the system, as an overall, can be determined generally based on the hardware.

Problem Definition

The problem to be solved in this project is a simulation based on the Lorentz force equation. The formula is given by:

$$F = q \cdot \vec{v}_q \times \vec{B} \tag{1}$$

In (1), q is the charge of the particle, $\overrightarrow{v_q}$ is the velocity of the particle, and $\vec{B}$ is the magnetic field that the particle is experiencing on a specific position. To obtain the motion of the particle, equation at (1) is related to Newton's second law of motion equation:

$$F = m \cdot \vec{a} \tag{2}$$

where:

$$\vec{a} = \frac{d\vec{v}}{dt} = \frac{(\overrightarrow{v_1} - \overrightarrow{v_0})}{\Delta t} \tag{3}$$

Assuming a particle is placed in an initial position in space $\overrightarrow{P_0}$, with an initial velocity $\overrightarrow{v_0}$, and under a magnetic field $\overrightarrow{B_0}$, the particle next position, and next velocity, can be calculated, provided that the magnetic field on that next position is known.

Proof of Concept

By combining equations in (1), (2) and (3), the following is obtained:

$$m \cdot \frac{(\overrightarrow{v_1} - \overrightarrow{v_0})}{\Delta t} = q\, \overrightarrow{v_0} \times \overrightarrow{B_0} \tag{4}$$

Then, by dividing the mass (m) on both sides of (4), the following is obtained:

$$\vec{a} = \frac{(\overrightarrow{v_1} - \overrightarrow{v_0})}{\Delta t} = \frac{q}{m} \cdot \overrightarrow{v_0} \times \overrightarrow{B_0} \tag{5}$$

And finally, by multiplying (5) by Δt on both sides the resultant equation is:

$$(\overrightarrow{v_1} - \overrightarrow{v_0}) = \frac{q}{m} \cdot (\overrightarrow{v_0} \times \overrightarrow{B_0}) \cdot \Delta t \tag{6}$$

Based on (6), the next velocity based on the initial can be expressed as:

$$\overrightarrow{v_1} = \frac{q}{m} \cdot (\overrightarrow{v_0} \times \overrightarrow{B_0}) \cdot \Delta t + \overrightarrow{v_0} \tag{7}$$

By determining $\vec{a}$ from (4) and replacing in (7), $\overrightarrow{v_1}$ can also be expressed as:

$$\overrightarrow{v_1} = \overrightarrow{a_0} \cdot \Delta t + \overrightarrow{v_0} \tag{8}$$

Knowing the initial position $\overrightarrow{P_0}$, velocity $\overrightarrow{v_0}$, time Δt, and acceleration a_0, the next position $\overrightarrow{P_1}$ can be obtained as:

$$\overrightarrow{P_1} = \overrightarrow{P_0} + \overrightarrow{v_0} \cdot \Delta t + \frac{1}{2} \cdot \overrightarrow{a_0} \cdot (\Delta t)^2 \tag{9}$$

For the time component it is important to understand that the movement of a charged particle, under a constant magnetic field, is in general a helix [3]. This type of motion follows a circular rotation toward one direction with a guiding center. Therefore, for the time component, a frequency of rotation is associated with the motion by:

$$\omega = \frac{q}{m} \cdot \vec{B} = \frac{2\pi}{t} \tag{10}$$

as a result :

$$t = \frac{2\pi \cdot m}{q \cdot \vec{B}} \tag{11}$$

Based on (11) it is possible to know the time it takes a charged particle to perform one rotation. However, in our case, the magnetic field is variable on each position. This implies that the time component is different on each position. To determine the next position, a small piece or fraction of the entire rotation time must be obtained. This can be expressed as:

$$\Delta t = xt = x \cdot \frac{2\pi \cdot m}{q \cdot \vec{B}} \tag{12}$$

where $x < 1$.

One important aspect of these equations is to understand their dependencies. In our case, (9) cannot be determined until $\overrightarrow{a_0}$ is determined in (4). And (4) cannot be determined until Δt is determined (12). This may be a challenge for the processing time, if not controlled by the program flow. However, in this case, all particles will follow the same fairness or sequence of calculations. This means the program unit will execute in a serial way for one particle, but a parallelism attempt is made for a greater amount of particles.

Problem Constraints

In order to perform the simulation, the following constraints have to be met:

- All vectors are represented with values for i, *j*, and *k*.
- $\overrightarrow{P_0}$ is a position vector with values at P_x, P_y, and P_z, whose magnitude is not zero (0).
- $\overrightarrow{v_0}$ is a velocity vector with values at v_x, v_y, and v_z, whose magnitude is not zero (0).
- $\overrightarrow{B_0}$ is the magnetic field present at the initial position ($\overrightarrow{P_0}$) with values at B_x, B_y, and B_z, whose magnitude is not zero (0). The magnetic field shall be present always.
- A particle *q* is an electric charge *e*, for instance, all particles have the same mass and charge given by:

  $m = 9.10938188 \times 10^{-31}$ Kg

  $q = 1.60217646 \times 10^{-19}$ C

- Total distance to calculate the trajectory is a cylinder with dimensions of 60 cm width for x, 25 cm height for y, and 25 cm depth for z.
- All units are in metric notation: N for Newton, C for coulombs, T for Teslas, A for Amperes, g for grams, and m for meters. Any of these units can be superseded by Kilo(k), Centi(c), Mili(m), Micro(μ), or Nano(n)
- Time is measured in seconds and Frequency in Hertz (1/s). And it is a fractional value, small enough, that allows for a higher precision.

All these constraints can be modified on future implementations for the same system.

Accuracy vs. Iterations

An important aspect of this problem is to understand the impact on the accuracy of the calculations. While the value for the time can be calculated as expressed in (12), the displacement of the particle has to be small enough to provide the accuracy of the motion. This implies that several calculations have to be made for (4) in a very small fraction of time to obtain:

$$\vec{a} = \lim_{t \to 0} \frac{d\vec{v}}{dt} \tag{13}$$

By using a very small fraction of the time, an entire non-linear motion is being represented with linear equations. In this case, the accuracy is highly dependent on the granularity of the values and the amount of iterations used to project just one rotation. Given the constraint on the distance to move, from 0 to 60cm in x, or from 0 to 25 cm in y or z, several iterations have to be made to obtain the displacement.

To provide an idea on the importance of the accuracy vs. amount of iterations, consider Figure 1. In this example, the followings are assumed: an electric charge with an initial velocity of (1, 0, 0) km/s, an initial position of (0, 0, 0) cm, and a constant magnetic field of (1, 0, 0) mT.

By using (12), (4) and (9), it can be observed that the fraction of time used for 10,000 iterations yields to different results for the position in *x*. For a fraction of time of 0.1 the particle has been displaced a total of 3.55 cm, very distant. While for a fraction of time of 0.001, the particle has been displaced only 0.356 mm, very close. The accuracy obtained by decreasing the fraction of time, as established in (13), while it provides more information it is also more detailed. This is an important concept to understand the computational power required to perform these simulations.

As more precision is required, the value for the fraction of time has to be decreased. This project considers operating in the range of $1x10^{-6}$ to $1x10^{-9}$.

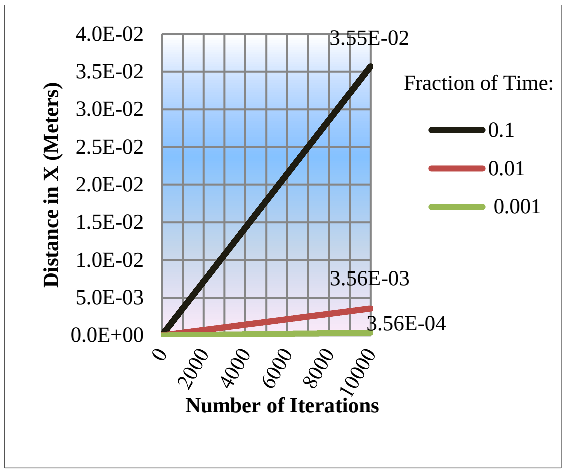


Figure 1
Distance in X for Different Fractions of Time

THE HIGH PERFORMANCE COMPUTING APPROACH

Today's processors are aimed to perform more tasks with less power consumption [4]. Most of the actual CPUs available on the market are multi core. In order to obtain the most out of these CPU, it is required a change in the programming approach [5]. While most schools today topics are for using visual development tools to exploit the usability of a system, performing computations to run simulations is other topic. Even though it is possible to run this simulation on a six core CPU, to accomplish that, new hardware and some parallelization on the program is required. To solve the problem, is possible to obtain a more precise result with large amount of iterations using a high performance computing cluster.

Hardware Availability

At the Polytechnic University of Puerto Rico, in the Computer Engineering Laboratory, a total of four high performance computers (HPC) clusters are available. Two of them are SGI proprietary processors, closed systems. Due to lack of support documentation, a decision was made to not work on those clusters. The other two clusters available are Intel/AMD based, running ROCKS cluster software Release 4.2.1. ROCKS is an optimized Linux for running on clusters to perform job schedules and parallelism. One of the clusters was updated from ROCKS 4.3 x64 to release 5.4 x64 bit. This cluster is based on DELL© PowerEdge servers, with the head node based on Intel© CPU, and 16 compute nodes based on AMD© CPU. This cluster provides simulations with MATLAB Distributed Computing Server R2011a. One of the options available is to provide the visualization of the output obtained from the simulation with MATLAB.

For the simulation project, the hardware platform consists of a high performance computer cluster based on DELL© PowerEdge servers, with one head node and 32 compute nodes. Each machine has two Intel Xeon Nocona processors of 2.8GHz. This processor is a single core/single thread 64 bit CPU, with 1MB L2 Cache, and 800 MHz FSB speed. Table 1 lists the basic specifications of this cluster.

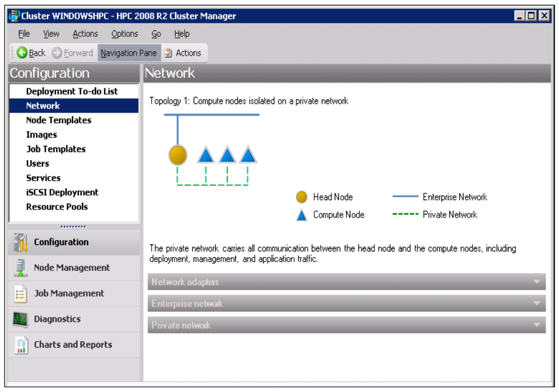
This cluster was totally changed and reconfigured, from ROCKS cluster software Release 4.2.1 x64, to Microsoft Windows HPC 2008R2. This is a 64 bit operating system that provides the optimization for managing a High Performance Computing environment plus the .Net framework for parallel task programming.

Windows HPC provides up to five Network Topology configurations [6]. As Figure 2 illustrate, the actual network configuration on the head node is Topology 1. On the Private network adapter, all compute nodes are connected through a dedicated SMC Tiger 48 ports switch at 1 Gbps and isolated from the public.

To provide access to the Windows HPC head node from the public network, the Enterprise network adapter is connected to a public switch at 100 Mbps. The head node is the only one that can contact, manage, and distribute all the jobs to the compute nodes.

Table 1
Windows Based HPC Cluster Hardware Specifications

Model	CPU	RAM	HD	Purpose
PowerEdge 1850	IntelXeon 2.8GHz	2 GB	SCSI 68.24GB	Head Node
PowerEdge SC1425	IntelXeon 2.8GHz	2 GB	ATA 37.25GB	Compute Node



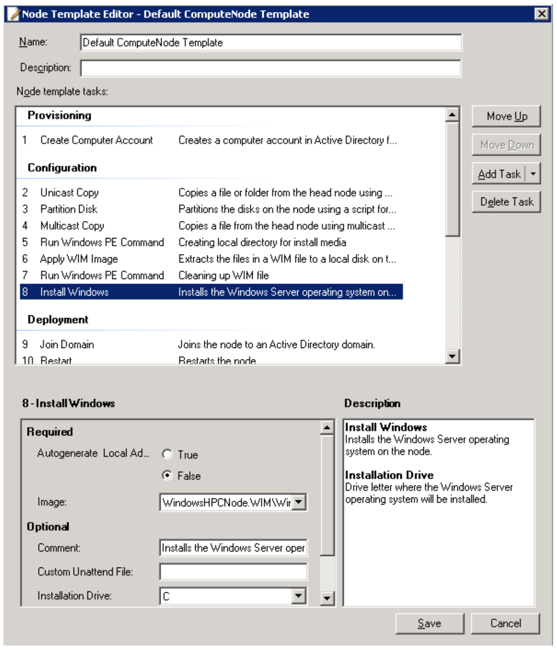
Topology 1: Compute nodes isolated on a private network

The private network carries all communication between the head node and the compute nodes, including deployment, management, and application traffic.

Figure 2
Network Configuration Topology

The HPC Operating System Deployment

Installing Windows HPC 2008R2 is very simple. The operating system has to be installed first on the Head Node. For the PowerEdge 1850, the driver for the DELL PERC 4e/Si RAID controller is not automatically recognized; it had to be loaded during the installation process. Once the head node is ready, as Figure 3 shows, a deployment template for the compute nodes has to be created. This template provides, among other things, the steps to install and configure the compute node.



Name: Default ComputeNode Template
Description:
Node template tasks:

Provisioning		
1	Create Computer Account	Creates a computer account in Active Directory f...
Configuration		
2	Unicast Copy	Copies a file or folder from the head node using ...
3	Partition Disk	Partitions the disks on the node using a script for...
4	Multicast Copy	Copies a file from the head node using multicast ...
5	Run Windows PE Command	Creating local directory for install media
6	Apply WIM Image	Extracts the files in a WIM file to a local disk on t...
7	Run Windows PE Command	Cleaning up WIM file
8	Install Windows	Installs the Windows Server operating system on...
Deployment		
9	Join Domain	Joins the node to an Active Directory domain.
10	Restart	Restarts the node

8 - Install Windows

Required
Autogenerate Local Ad... True [✓ False]
Image: WindowsHPCNode.WIM\Wi
Optional
Comment: Installs the Windows Server oper
Custom Unattend File:
Installation Drive: C

Description
Install Windows
Installs the Windows Server operating system on the node.
Installation Drive
Drive letter where the Windows Server operating system will be installed.

Figure 3
Default Deployment Template for the Compute Nodes

In order to start the deployment process, it is enough to turn on the compute node and boot it from the network via PXE, while the Head Node accepts all PXE requests.

With this deployment approach there are some disadvantages. Every time the compute node is rebooted, the template will be reapplied to the same hardware. This implies that the head node will automatically generate and assign a new node name, a new IP address, and reinstall everything from the beginning on the compute node.

To avoid this, Windows HPC provides to proactively add the compute nodes through an import of an XML file definition. The file basically contains the Machine GUID, and MAC Address. This guarantees a persistent deployment whenever the same hardware is rebooted. The template will be reapplied with the same computer name, IP address, and remaining configuration for the same hardware.

Although it is possible to install an alternate job scheduler, the native Windows job scheduler engine is the preferred choice for this project. Microsoft job scheduler engine provides for two types of tasks, basic task, and parametric tasks. A basic task uses a single command line that includes the command to run, along with the metadata that describes how to run the command. While a parametric task contains a command line with wildcards, allowing the same task to be run many times with different inputs for each step [7].

The Software Component

Using Windows HPC as the Operating System platform allows running .Net programming on the Windows HPC cluster. The programming language used in this project is C#, which allows for integration with MPI [8] and with the Windows Task Parallel Library for .Net.

The Windows Task Parallel Library is a new set of class library types, available to simplify parallel development [9]. It allows for parallel code to be written without having to work directly with threads or thread pools.

Algorithm Description

The target of the program is to provide the position of several charge particles. These particles are free electrons at an initial position, and with an initial velocity. For each particle, the positions and velocities are generated uniformly or randomly and stored on a single file to be used as input to the program. This file is as big as the amount of particles required to simulate.

The Magnetic field for the surroundings is provided by a separate class that calculates the magnetic field at any given point. The Magnetic field follows a Gaussian distribution for a cylinder with two circular loops on each end of the cylinder. This is equivalent to the Plasma Chamber available at the Plasma Research Laboratory at Polytechnic University of P.R.

As illustrated in Figure 4, for each particle's position, the Magnetic Field is obtained from an external class. This is used to determine Δt by using (12) with a predefined scaling factor for *x*. Afterward, the program calculates (5), (8), and finally (9) to obtain the next position. This position is provided again to the Magnetic Field external class, and the cycle repeats until the maximum amount of iterations are reached. These steps for one particle, is programmed to execute on one single CPU thread.

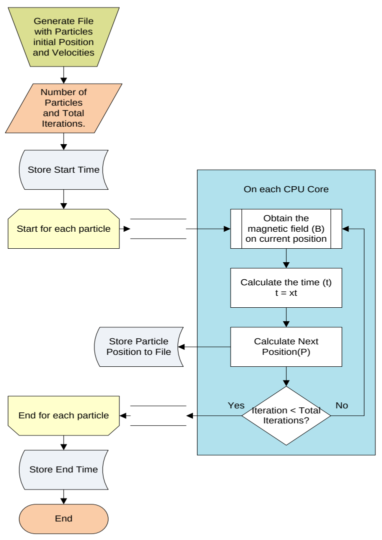

Generate File with Particles initial Position and Velocities
Number of Particles and Total Iterations.
Store Start Time
Start for each particle
On each CPU Core
Obtain the magnetic field (B) on current position
Calculate the time (t) t = xt
Calculate Next Position(P)
Store Particle Position to File
Yes
No
Iteration < Total Iterations?
End for each particle
Store End Time
End

Figure 4
Basic Program Flow

Parallelism Technique

The file generated with the particles position and velocities is shared among all compute nodes. Windows HPC provides a shared repository for such scenarios.

Since each compute node has only two CPU, when two particles are analyzed, it is expected that one goes to one CPU and the other particle goes to the second CPU. If more than two particles are needed to be analyzed by a single node, then each particle will follow the assigned pipeline by the compute node internal CPU control. At the moment of submitting the job, the particles shall be equally distributed among all the available compute nodes. Under some circumstance, where no equal distribution can occur, some nodes may have more particles than others.

The program stores on a distinct file the start and end time it took for analyzing a single particle. This will help on doing the comparison for the testing scenarios.

TESTING SCENARIOS

For testing purpose of this project, the same program is written in three different formats:

- Format 1: simple program with no parallelism libraries incorporated or multithread calls.
- Format 2: program with MPI library calls.
- Format 3: program with Windows Task Parallel Libraries calls.

To determine the best approach for the simulation, a comparison in performance between the different distribute computing techniques is required. The following steps are performed to help on the comparison:

- Determine the amount of available compute nodes and find out the total amount of particles to process by multiplying the amount of compute nodes by 4.
- Generate the initial positions and velocities file, and place it in the shared repository.
- Send the Format 1 program, with no parallelism implemented, to a single node with

all particles. This will be called the baseline for all future comparisons.

- Send the Format 2 program with MPI to all available nodes.
- Send the Format 3 program to all available nodes, in such a way that each node will calculate for 4 particles. This will require MPI plus Task Parallel Library.

VISUALIZATION OF THE SIMULATION

Although it is not a requirement, a simulation is not complete without a visualization component. The important part of the simulation is to take decisions or learn something new based on the output. For the visualization part, two options are available: MATLAB or PARAVIEW. In either case, the output of the program has to be formatted to be processed by the preferred tool.

Proposed Solution with ParaView

ParaView is an open-source, multi-platform application for the visualization and analysis of scientific datasets [10]. The advantage of using ParaView is the easy to use interface and the powerful filter capabilities it provides. Also, it can be run on a client-server environment. In addition, ParaView can open files from several formats. To install ParaView on the Windows HPC cluster head node was very simple. However, to allow for parallel processing of the visualization, ParaView needs to be installed on each compute node as well.

Alternate Solution with MATLAB

The advantage of using MATLAB is that it runs on a separate cluster, so the Windows HPC cluster can be performing new calculations, and generating a new file, while the MATLAB cluster is processing the file for visualization. The shortcoming to this approach relies on the file transfer due to the file size, however, the communication between the Windows HPC cluster and the MATLAB cluster is via the Public Network at 1 Gbps. Once the file is in the MATLAB cluster it can be processed using the proper MATLAB visualization toolbox from a client PC.

PROJECT RESULTS

From the testing scenarios, a total of 20 nodes were used. The graph on Figure 5 illustrates the results on execution time for each processing technique used for the same simulation. The execution for the single node was the slowest, taking a considerable amount of time to complete execution. The execution with the MPI was the faster, while the execution with MPI + Task Parallel Library was slower than MPI alone.

CONCLUSIONS AND FUTURE PLANS

By comparing MPI with Task Parallel Library combined with MPI, we can conclude that MPI is faster for this hardware implementation. This project provides the baseline for further research in Plasma Physics. It allows also, to other graduate or undergraduate future candidates, to integrate other conditions into the program, like for example analyzing the addition of an Electrical Field variable to the equation. It is possible that such alterations change the results in performance obtained in this paper.

For future works, the MATLAB cluster can be used to provide the visualization part of the simulation. Alternatively, Paraview also can be installed on the MATLAB cluster as well.

Other conditions, like adding more nodes can be evaluated.

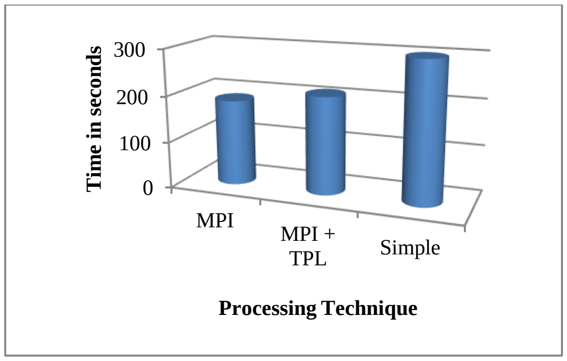


Figure 5
Processing Technique vs. Time in Seconds

ACKNOWLEDGEMENTS

This work was supported in part by the U.S. Army Research Office under grant W911NF-11-1-0180.

REFERENCES

[1] Bartoš, P. P.,et al. “Hybrid computer simulations: electrical charging of dust particles in low-temperature plasma”, *European Physical Journal* D 54(2), 2010, p319-323.

[2] Sommerville, Ian, “”,*Software Engineering* ,8th ed., Addison-Wesley, 2007, p

[3] Chen, Francis, F. , “Chapter 2, Single Particle Motions”, *Introduction to Plasma Physics and Controlled Fusion* , 2nd ed. Vol. No. 1, Springer, 1984, p19-49

[4] Koomey, Jonathan, G., et al.”Assessing Trends in the Electrical Efficiency of Computation Over Time”, *IEEE Annals of the History of Computing* ,August 5, 2009

[5] Asanovic, Krste, et al. “A View of the Parallel Computing Landscape”, *Communications of the ACM*, Vol. No. 52 Issue 10, October 2009, p56-67

[6] Microsoft Corp., ”Windows HPC Server 2008 R2”, Appendix 1 – HPC Cluster Networking, *Microsoft TechNet Library*, May 2012 http://technet.microsoft.com/en-us/library/ff919486(v=ws.10).aspx

[7] Microsoft Corp., “Parallel Programming in the .NET Framework”, .Net Framework 4, *Microsoft TechNet Library*, May 2012 http://msdn.microsoft.com/en–us /library/dd460693.aspx

[8] Robison, A, et al. “Using MPI with C# and the Common Language Infrastructure”, Technical Report TR570, Indiana University, School of Informatics and Computing, Oct. 2002

[9] Microsoft Corp., “Task Parallel Library”, .Net Framework 4, *Microsoft TechNet Library,* May 2012 http://msdn. microsoft.com/en-us/library/dd460717.aspx

[10] Kitware, Inc., ParaView User’s Guide (V3.14), May 2012 http://www.itk.org/Wiki/ParaView