Optimized simulations of particle interactions in controlled nuclear fusion research

---

Written Report

2021 Group 11-12

Members:

Alex Pan 3i120

Dylan Lee 3i208

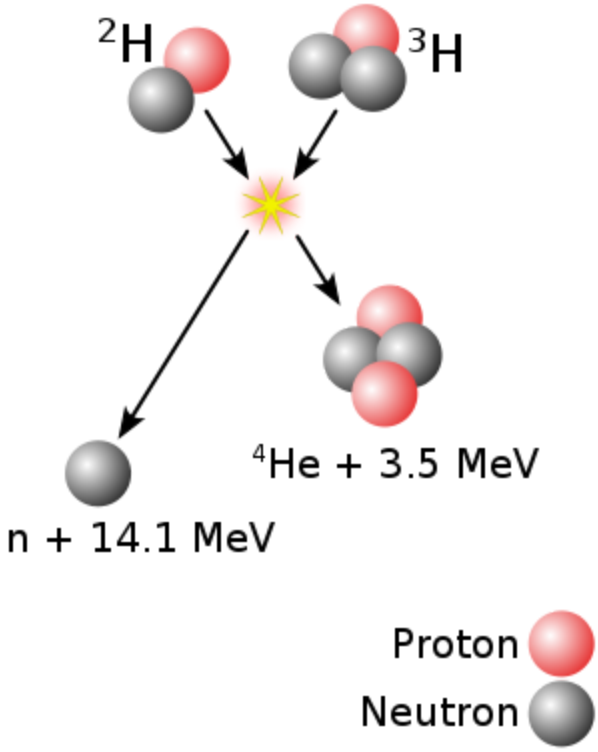
Emmanuel Zeth Liang 3i209

Abstract

With the advancement of nuclear fusion research, it has become the focus of 21st century technology due to the possibilities it brings for energy. However, due to the costly nature of this it has become especially imperative for scientific simulations of plasma movement. This project seeks to integrate available technologies with our current computing and scientific knowledge to simulate particle motion in specific fields.

Introduction

Nuclear fusion refers specifically to the fusion of two or more nuclei of atomic particles to form one or more different atomic nuclei and subatomic particles as well as high amounts of energy that is released or absorbed, due to the difference in mass of the reactants and products of nuclear fusion (ITER, n.d.). In most experiments, researchers attempt to fuse hydrogen and deuterium because the reaction only produces helium and a neutron, which are relatively harmless compared to the large amounts of radioactive waste produced by the nuclear fission of Uranium-235, the isotope of uranium used most often in nuclear fission power plants today (Department of Energy, n.d.). The fusion of atoms in a controlled manner can theoretically produce almost 4 million times more energy than the chemical reaction of burning coal, oil, gas etc (Parfit, 2021). It also produces 4 times more energy than nuclear fission, with the production of far less radioactive waste.



Plasma physics is the crux of nuclear fusion. There are three conditions that are crucial for fusion to be achieved in a laboratory: extremely high temperatures, sufficient plasma particle density, as well as sufficient confinement. High temperature translates to a high amount of kinetic energy to provoke high energy collisions of particles, while high particle density increases the probability of such collisions from occurring, and sufficient confinement time to hold the plasma at a fixed volume (ITER, n.d.). Plasma, in layman terms, is essentially the driving force of nuclear fusion, allowing for the fusion of particles in a magnetic confinement environment. Therefore, to bring the concept of nuclear fusion to reality, one must understand plasma physics.

Plasma refers to the fourth state of matter, a superheated ionized gaseous substance that consists of positively charged ions, electrons and neutrals. What defines plasma is the fact that its properties are dominated by electric and magnetic fields, and that is quasi-neutral, meaning that the densities of its negative and positive charges are approximately the same (Wiesemann, 2014). Plasma is defined by the ion density, electron density and its neutral density. It also has

various properties, including quasi-neutrality and ionization degrees, plasma oscillations, its behaviour as a gas, particle mobility and conductivity, diffusion in plasma, the sheath formation in plasma. Of paramount importance is the movement of particles in plasma under the influence of electric and magnetic fields. The motion of particles in the plasma is denoted by the formula of $ma = qvB + F$. From here, the frequency of the cyclotron motion of a charged particle can be given as: $\omega_B = \frac{V_\perp}{r_B} = \frac{qB}{m}$. Its magnetic moment is given as $M = I \cdot A = \frac{W_\perp}{B}$, where W is given as kinetic energy due to the gyration of the charged particle. We also know that the value of W is a constant of motion. Other unique properties of plasma are its AC conductivity when magnetized, waves in plasma with and without an external magnetic field, the fluid description of plasma, and the general dispersion relation of cold magnetized plasma.

Since we are living in the 21st century, we can make use of relatively novel technologies that can assist researchers in the field of nuclear fusion and plasma research. An example of these technologies would be computer visualizations as well as artificial intelligence that makes use of machine learning to avoid problems in magnetic confinement units. Researchers have made use of visualization machines such as OpenGL to visualize magnetic field data from the DIII-D Tokamak extremely efficiently (Schussman et al, 2000). This visualization allows scientists to have a clearer communication regarding magnetic field structure in the DIII-D Tokamak, which has allowed them to have a deeper understanding regarding the data collected from experiments conducted in the Tokamak, accelerating their research. The researchers made use of numerical simulations using the geometry of the Tokamak to help them calculate the topology of the magnetic flux linkage inside the high temperature fusion plasma. The researchers then visualized the data collected in 3 dimensions and 2 dimensions. This visualization then helped the scientists further understand certain phenomena in the Tokamak. For example, of particular interest to the researchers are magnetic islands, which are closed magnetic flux tubes that can have detrimental effects on energy confinement in the high temperature plasma of the DIII-D Tokamak. As such, the visualization of magnetic flux linkage in the Tokamak helped scientists to combat these problems and deepen their understanding of plasma physics and nuclear fusion in magnetic confinement fusion research.

Visualization of particle movement can be achieved, using modern-day visualization softwares such as MATLAB. There is a widely known method of setting up the visualization of the numerical trajectory of a particle in a static magnetic field by making use of Newton's second law in the form of $F = m \cdot \frac{dv}{dt}$, as well as the Lorentz equation in the form of $F = e(E + v \cdot B)$, where F represents resultant force, m is the mass of particle, dv and dt represent the change in velocity and change in time respectively, e represents the charge of the particle, E represents the external electric field, v represents the velocity of the particle, and B is the magnetic field density (Michel, n.d.). Assuming planar motion in a uniform magnetic field in the X-Y dimensions, from these two equations can be derived $V_y' = V_y - V_x P$ as well as $V_x' = V_x - V_y P$ making use of the numerical method of integration, where V' represents the velocity after 1 time step and V represents the velocity before 1 time step. P represents the dimensionless constant of $\frac{eB_0 \Delta(t)}{m}$. These two simple equations can then be reiterated in the form of $x' = x + v_x \Delta(t)$ and $y' = y + v_y \Delta(t)$. However, the set of equations produced do not produce the desired result in a simulation program as while it does iterate the expected circular motion of the particle in the magnetic field, the energy is not conserved as is required by the differential equations above and hence is inaccurate at larger timescales.

Researchers from Princeton have made use of machine learning in the form of artificial intelligence to assist them in the control and avoidance of disruptions in tokamaks that cause plasma currents to fall rapidly to zero and result in unnecessary maintenance (Fu et al, 2020). Normally, large tokamaks such as TEXTOR (Tokamak Experiment for Technology Oriented Research) have used physics-based models to detect disruption precursors (TEXTOR, n.d.), an example being making use of heterodyne electron cyclotron emission diagnostic to detect an m/n = 2/1 disruption (Jeong et al, 2003). However, the complexity of these disruptions has led to more sophisticated disruption prediction methods being developed. Although some of these prediction methods are still physics based models, the researchers focused on the usage of machine learning algorithms to recognize precursors to disruptions. Thus, the researchers came

up with a disruption prediction model and algorithm for the DIII-D. Their results showed that disruption and tearing mode prediction could have a window of around 250ms before the event occurred with a high rate of positive disruption detection of around 90% and a relatively small false disruption detection, thus proving machine learning being used in disruption detection.

Another large aspect of the project was the optimisation of the code. Code optimization is any method of code modification to improve code quality and efficiency. A program may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer input/output operations. The conservative global gyrokinetic toroidal full-f five-dimensional Vlasov simulation (GT5D) is a nuclear fusion simulation program designed to analyze turbulence phenomena in tokamak plasma. (Norihisa et al) Researchers optimized it for graphics processing unit (GPU) clusters with multiple GPUs on each node. Based on the profile results of a GT5D on a CPU node, it was decided to offload the entire time development part of the program to GPUs, except for MPI communication. Their evaluation results show they achieved a maximum 3.35 times faster performance with a GPU during a function level execution, and 1.91 times faster total performance, than could be achieved via CPU-only execution, both in measurements on high density GPU cluster HA-PACS, where each computation node consists of four NVIDIA M2090 GPUs and two Intel Xeon E5-2670 (SandyBridge) that provide 16 cores in total. These performance improvements for a single GPU were obtained in measurements against four CPU cores, not a single-core CPU, and include a 63% performance gain obtained by communications overlapping between MPI processes and GPU calculations.

The aim of this project was thus to optimize simulations of particle interactions in controlled nuclear fusion research, so as to allow for the increase in efficiency that is required for further progress in nuclear fusion research. By integrating 21st century technology and computing together with our physics knowledge, we can hopefully achieve an improvement in the efficiency of the research conducted.

Methodology

We made use of two main softwares to conduct this project, based on recommendations from our external mentor, Code::Blocks as well as Paraview.

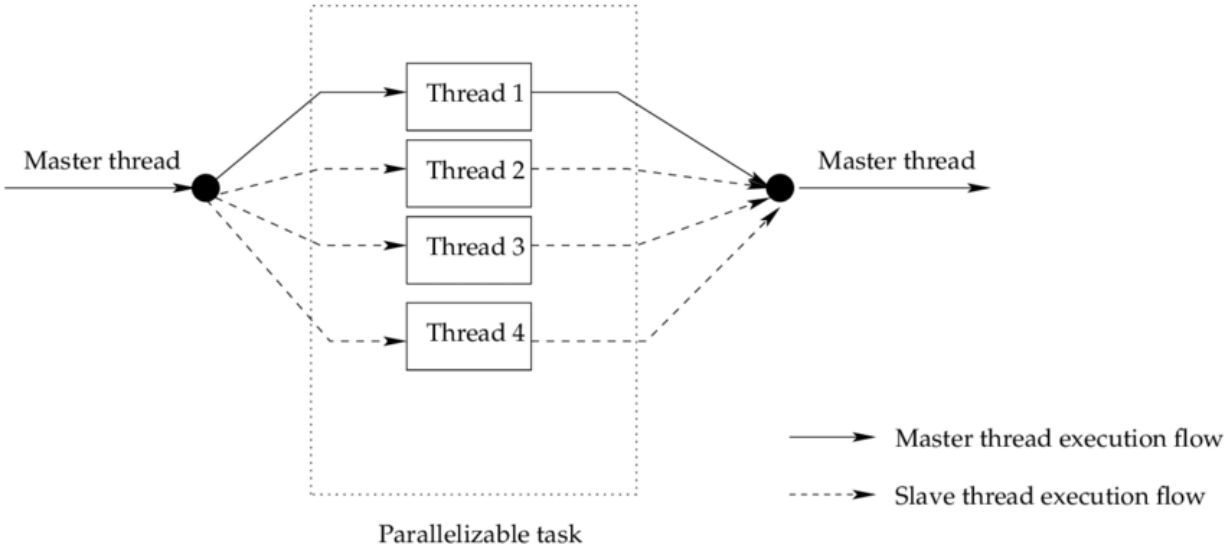
Code::Blocks is a free and open source C/C++ IDE, providing advantages such as a fast and convenient custom build system which enables the importing and inclusion of C++ libraries that bring additional useful functions to the code. Code::Blocks also has access to the GNU GCC Compiler, a compiler that supports optimisation features which are relevant to the project. With regards to the C++ libraries used, we made use of the GSL library (GNU Scientific Library), a software library for numerical computations in applied mathematics. In our case, GSL was used as a tool for random number generation. Another C++ library we utilised is the Chrono library, serving as a method to accurately keep track of the runtime of the code. This data plays an important role when evaluating the effectiveness of runtime optimization later on.

Paraview, on the other hand, is a free and open source multi-platform application used for interactive scientific visualization. With its plethora of features and filters, we are able to represent the particles and their associated trajectories in 3D. The code outputs a CSV (comma-separated values) file that can be read as a spreadsheet containing all the relevant variables such as the X, Y, and Z coordinates of each particle. In Paraview, the software uses a CSV reader to interpret the data, before the filters *TableToPoints* and *TemporalParticlesToPathlines* are applied to actually visualize the particle movement.
pic

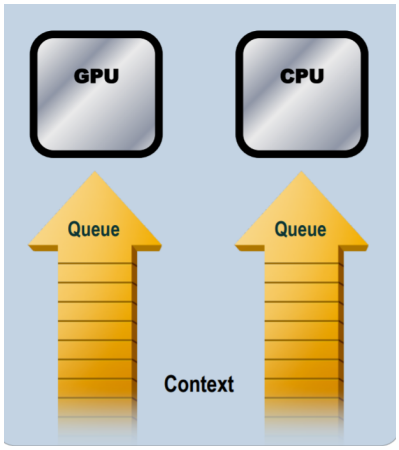
Hence, through Code::Blocks and Paraview, we undertook the task of compiling and visualising several iterations of code that demonstrate 2 separate scenarios of particle motion:

1. Electrons and/or protons and/or deuterons in a uniform magnetic field
2. Electrons and/or protons and/or deuterons in a uniform electric field

The next proposed step is to conduct optimization on our code through the implementation of OpenMP (Open Multi-Processing) and OpenCL (Open Computing Language). OpenMP introduces the aspect of parallelization to the code, which refers to the execution of blocks of code by multiple "threads" simultaneously. A master or primary thread represents the original series of code to be executed sequentially. At certain sections of the code, this thread forks out into several other threads that subdivide a given task among themselves and converge after a specified goal is achieved. Therefore, instead of waiting for a time-consuming or computationally intensive process to fully execute, OpenMP can expedite its completion with the help of multithreading. Separately, OpenCL is a framework that allows users to write code that is executable on other hardware processors or compute devices such as the CPU and GPU to facilitate parallel computing. A significant feature is the concept of kernels, which essentially replace traditional C++ loops with a function that executes a task at a given point. A command queue system submits commands like kernel executions and memory management to a specific device. As a result, with CPUs and GPUs possessing many components that can simultaneously run kernel executions, OpenCL can also reduce runtime.



Multithreading / Parallelisation Visualization



OpenCL Queueing Process Visualization

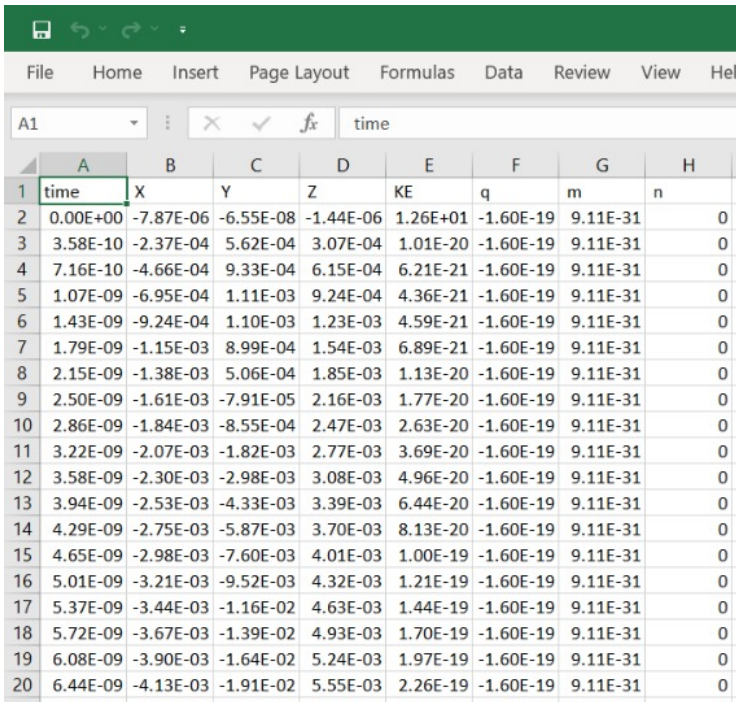
Code for Uniform fields (No particle interactions)

```
1  #include <iostream>
2  #include <fstream>
3  #include <chrono>
4  #include <cmath>
5  #include <gsl/gsl_rng.h>
6  #include <gsl/gsl_randist.h>
7  using namespace std;
8  int ndatapoints=1000;
9  int ncalc=1836;
10 //No. protons, electrons and deuterons
11 int pro=0;
12 int elec=1;
13 int deu=0;
14 int npart= pro+elec+deu;
15 int main()
16 {
17     std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::no
18     //declare and allocate memory for parameters of each particle
19     double *time=new double[npart];
20     double *x0=new double[npart];
21     double *y0=new double[npart];
22     double *z0=new double[npart];
23     double *x1=new double [npart];
24     double *y1=new double [npart];
25     double *z1=new double[npart];
26     double *vx=new double[npart];
27     double *vy=new double[npart];
28     double *vz=new double[npart];
29     double *q=new double[npart];
30     double *m=new double[npart];
31     double *KE=new double[npart];
32     double *EXx=new double[npart];
33     double *EXy=new double[npart];
34     double *EXz=new double[npart];
35     double determinant;
36     double dt=(1/((1.6e-19)*0.01/(2*3.142*9.11e-31)))/100;
37     cout<< dt<< endl;
```

At the beginning of the code, all the libraries which are needed are included. Lines 19 to 34 demonstrate the use of dynamic memory to initialize variables, since the programme is likely to handle a large number of particles that will potentially exceed the compiler allocated memory limit, dynamic memory allocation allows for the allocation of variable memory size to each variable in the form of an array of length *npart*. Line 17 shows the starting of the Chrono library's time-keeping clock.

```
ofstream o_file;
o_file.open ("umf.csv");
o_file << std::scientific;
o_file.precision(4);
o_file <<"time, X, Y, Z, KE, q, m, n"<<endl;
```

This block of code prepares a CSV file with the column headers of variables which are needed to accurately represent particle motion in Paraview later on. A typical CSV file appears as such below.



File Home Insert Page Layout Formulas Data Review View Hel

A1 | time

	A	B	C	D	E	F	G	H
1	time	X	Y	Z	KE	q	m	n
2	0.00E+00	-7.87E-06	-6.55E-08	-1.44E-06	1.26E+01	-1.60E-19	9.11E-31	0
3	3.58E-10	-2.37E-04	5.62E-04	3.07E-04	1.01E-20	-1.60E-19	9.11E-31	0
4	7.16E-10	-4.66E-04	9.33E-04	6.15E-04	6.21E-21	-1.60E-19	9.11E-31	0
5	1.07E-09	-6.95E-04	1.11E-03	9.24E-04	4.36E-21	-1.60E-19	9.11E-31	0
6	1.43E-09	-9.24E-04	1.10E-03	1.23E-03	4.59E-21	-1.60E-19	9.11E-31	0
7	1.79E-09	-1.15E-03	8.99E-04	1.54E-03	6.89E-21	-1.60E-19	9.11E-31	0
8	2.15E-09	-1.38E-03	5.06E-04	1.85E-03	1.13E-20	-1.60E-19	9.11E-31	0
9	2.50E-09	-1.61E-03	-7.91E-05	2.16E-03	1.77E-20	-1.60E-19	9.11E-31	0
10	2.86E-09	-1.84E-03	-8.55E-04	2.47E-03	2.63E-20	-1.60E-19	9.11E-31	0
11	3.22E-09	-2.07E-03	-1.82E-03	2.77E-03	3.69E-20	-1.60E-19	9.11E-31	0
12	3.58E-09	-2.30E-03	-2.98E-03	3.08E-03	4.96E-20	-1.60E-19	9.11E-31	0
13	3.94E-09	-2.53E-03	-4.33E-03	3.39E-03	6.44E-20	-1.60E-19	9.11E-31	0
14	4.29E-09	-2.75E-03	-5.87E-03	3.70E-03	8.13E-20	-1.60E-19	9.11E-31	0
15	4.65E-09	-2.98E-03	-7.60E-03	4.01E-03	1.00E-19	-1.60E-19	9.11E-31	0
16	5.01E-09	-3.21E-03	-9.52E-03	4.32E-03	1.21E-19	-1.60E-19	9.11E-31	0
17	5.37E-09	-3.44E-03	-1.16E-02	4.63E-03	1.44E-19	-1.60E-19	9.11E-31	0
18	5.72E-09	-3.67E-03	-1.39E-02	4.93E-03	1.70E-19	-1.60E-19	9.11E-31	0
19	6.08E-09	-3.90E-03	-1.64E-02	5.24E-03	1.97E-19	-1.60E-19	9.11E-31	0
20	6.44E-09	-4.13E-03	-1.91E-02	5.55E-03	2.26E-19	-1.60E-19	9.11E-31	0

```
for(int n=0; n<npart; n++)
{
    time[n]=0;
    x0[n]=gsl_ran_flat(rng,-xL,xL);
    y0[n]=gsl_ran_flat(rng,-yL,yL);
    z0[n]=gsl_ran_flat(rng,-zL,zL);
    if(n<pro){ //protons
        q[n]=1.6e-19;
        m[n]=1.67e-27;
    }
    else if( n>=pro && n<(elec+pro) ){ //electrons
        q[n]=-1.6e-19;
        m[n]=9.11e-31;
    }
    else if(n>=(elec+pro)){ //deutrerons
        q[n]=1.6e-19;
        m[n]=3.34e-27;
    }
    double velvar;
    velvar = sqrt((2*sigma*(1.6e-19))/m[n]);
    vx[n]=gsl_ran_gaussian(rng,velvar);
    vy[n]=gsl_ran_gaussian(rng,velvar);
    vz[n]=gsl_ran_gaussian(rng,velvar);
    xl[n]=vx[n]*dt+x0[n];
    yl[n]=vy[n]*dt+y0[n];
    zl[n]=vz[n]*dt+z0[n];
}
```

The for loop above iterates through all *npart* particles, assigning each particle a randomised starting coordinate and velocity using GSL functions. Using if/else statements, the for loop is able to distinguish between protons, electrons and deuterons and assign them the respective charges and masses. Throughout the code for all scenarios, the above processes are utilised.

When coding for particle motion in uniform electric and uniform magnetic fields, the conventional approach is to make use of Newton's second law in the form of $F = m \cdot \frac{dv}{dt}$, as well as the Lorentz equation in the form of $F = e(E + v \cdot B)$, where F represents resultant force, m is the mass of particle, dv and dt represent the change in velocity and change in time respectively, e represents the charge of the particle, E represents the external electric field, v represents the velocity of the particle, and B is the magnetic field density (Michel,

n.d.). Assuming planar motion in a uniform magnetic field in the X-Y dimensions, from these two equations can be derived $V_y' = V_y - V_x P$ as well as $V_x' = V_x - V_y P$ making use of the numerical method of integration, where V' represents the velocity after 1 time step and V represents the velocity before 1 time step. P represents the dimensionless constant of $\frac{eB_0 \Delta(t)}{m}$. These two simple equations can then be reiterated in the form of $x' = x + v_x \Delta(t)$ and $y' = y + v_y \Delta(t)$. However, the set of equations produced do not produce the desired result in a simulation program as it does not iterate the expected circular motion of the particle in the magnetic field, and the energy is not conserved as is required by the differential equations above (Michel, nd) for a uniform magnetic field.

Thus, for our simulation, we used the conventional method for effect of electric field, where there is no energy disparity, and in order to have proper conservation of energy, for magnetic interaction we implemented in C++ a method proposed by F.Michel to calculate particle movement when affected by a magnetic field, which involved the use of matrices. In the method. The Lorentz forces equations in 3 dimensions are represented in matrix form:

$$\begin{pmatrix} 1 & -p_z & p_y \\ p_z & 1 & -p_x \\ -p_y & p_x & 1 \end{pmatrix} \begin{pmatrix} v_x' \\ v_y' \\ v_z' \end{pmatrix} = \begin{pmatrix} 1 & p_z & -p_y \\ -p_z & 1 & p_x \\ p_y & -p_x & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

Where v_x' is the velocity of the particle in the next time step in the x-direction and vice versa for v_y' and v_z'.

An asymmetric matrix A can be defined such that :

$$A = \begin{pmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{pmatrix}$$

Where I is the 3 × 3 unit matrix and V and V' are 3 × 1 vectors.

The equation can then be written as $(I + A)V' = (I - A)V$, and so $V' = (I + A)^{-1}(I - A)V = BV$, hence allowing us to iterate this for every time step and find the velocities and thus the resulting displacement. Following this, we would explain how we implemented this into our C++ simulation code.

```
107                 double Ex,Ey,Ez,Bx,By,Bz;
108                 double dx,dy,dz;
109                 double a,b,c;
110                 Ex=0;
111                 Ey=0;
112                 Ez=0;
113                 Bx=0;
114                 By=0.01;
115                 Bz=0;
116                 time[n]=time[n]+dt;
117                 //Electric Field implementation
118                 EXx[n]= (q[n]/(2*m[n]))*Ex*dt*dt;
119                 EXy[n]= (q[n]/(2*m[n]))*Ey*dt*dt;
120                 EXz[n]= (q[n]/(2*m[n]))*Ez*dt*dt;
121                 //Magnetic field(matrix stuff)
122                 a=q[n]*Bx*dt/m[n];
123                 b=q[n]*By*dt/m[n];
124                 c=q[n]*Bz*dt/m[n];
125                 dx=2*xl[n]-x0[n]-c*y0[n]+b*z0[n];
126                 dy=2*yl[n]-y0[n]-a*z0[n]+c*x0[n];
127                 dz=2*zl[n]-z0[n]-b*x0[n]+a*y0[n];
128
129                 //Matrix inversion
130
131                 determinant = a*a + b*b + c*c + 1;
132                 double MI11, MI12, MI13, MI21, MI22, MI23, MI31, MI32, MI33;
133                 MI11 = 1+a*a;
134                 MI12 = a*b+c;
135                 MI13 = a*c-b;
136                 MI21 = (a*b)-c;
137                 MI22 = 1+b*b;
138                 MI23 = a+b*c;
139                 MI31 = a*c+b;
140                 MI32 = (b*c)-a;
141                 MI33 = 1+c*c;
142
143                 x0[n] = xl[n];
144                 y0[n] = yl[n];
145                 z0[n] = zl[n];
146
147                 xl[n] = ((MI11*dx+MI12*dy+MI13*dz) / determinant)+EXx[n];
148                 yl[n] = ((MI21*dx+MI22*dy+MI23*dz) / determinant)+EXy[n];
149                 zl[n] = ((MI31*dx+MI32*dy+MI33*dz) / determinant)+EXz[n];
```

Ex, Ey, Ez, Bx, By, Bz are the magnitudes of the electric and magnetic fields respectively in the x, y and z directions respectively. a, b and c are the p values in the x, y and z directions respectively and are calculated in lines 123-125 using the equations mentioned earlier. The matrix equation from the method above is then solved in the code from lines 123-149, getting the new position of the particle as x1[n] based on the magnetic field. In regards to the electric field, EXx, EXy and EXz are the effect the electric field has on the particle's displacement in the x, y and z directions respectively. EXx, EXy and EXz are derived from the Lorentz equation, $F = qE + qv \times B$, Newton's Second Law, $F = ma$ and equations of motion

$x_1 = x_0 + \frac{1}{2}(u + v)\Delta(t)$ and $v = u + a\Delta(t)$, u being the velocity at that time step and v being the velocity at the next timestep, x_1 being the particle's displacement at the next time step and x_0 being the particle's displacement at its current time step. The equations simplify to $x_1 = x_0 + u\Delta(t) + \frac{(qv \times B)(\Delta(t))^2}{2m} + \frac{qE(\Delta(t))^2}{2m}$. The matrix method already incorporates the entire equation barring the term containing E into the displacement and position calculation every time step, thus the electric field's contribution to the displacement of the particle is $\frac{qE(\Delta(t))^2}{2m}$, which EXx, EXy and EXz are the component vectors of. These calculations are therefore made in lines 118-120, and incorporated into the displacement calculation in lines 147-149. Thus, the effects of uniform magnetic field and/or uniform electric field can be accurately simulated, adhering to the laws of conservation of energy and giving the correct motions. The below diagrams demonstrate as such.

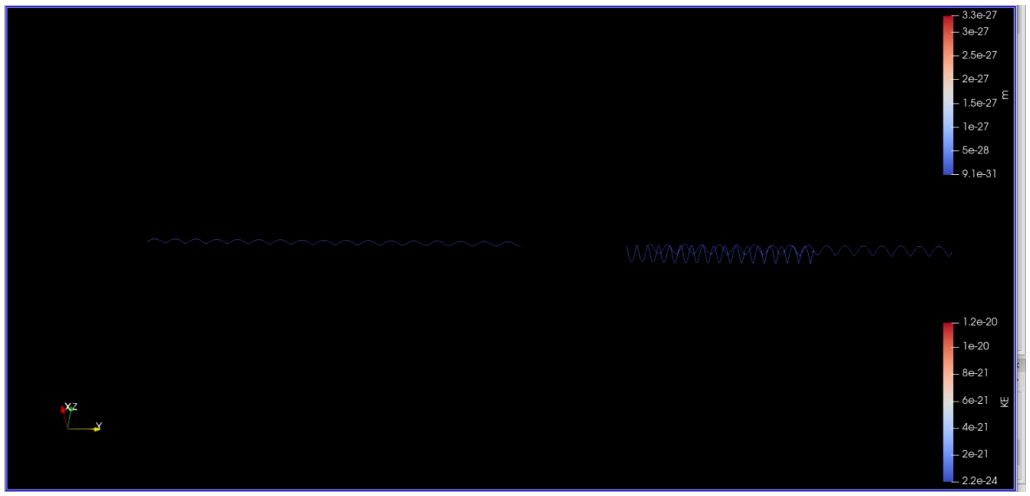


Fig 1.1 Scenario 1, electrons (side view)

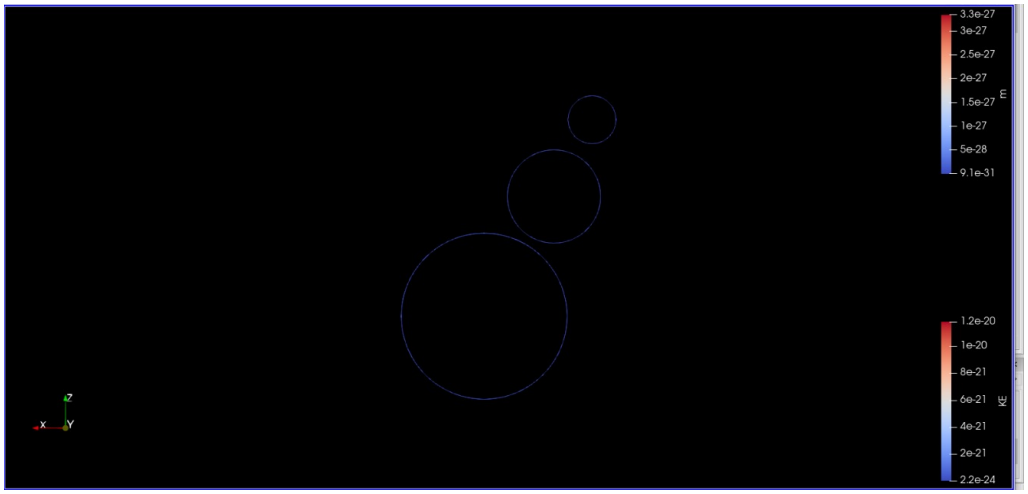


Fig 1.2, Scenario 1, electrons (front view)

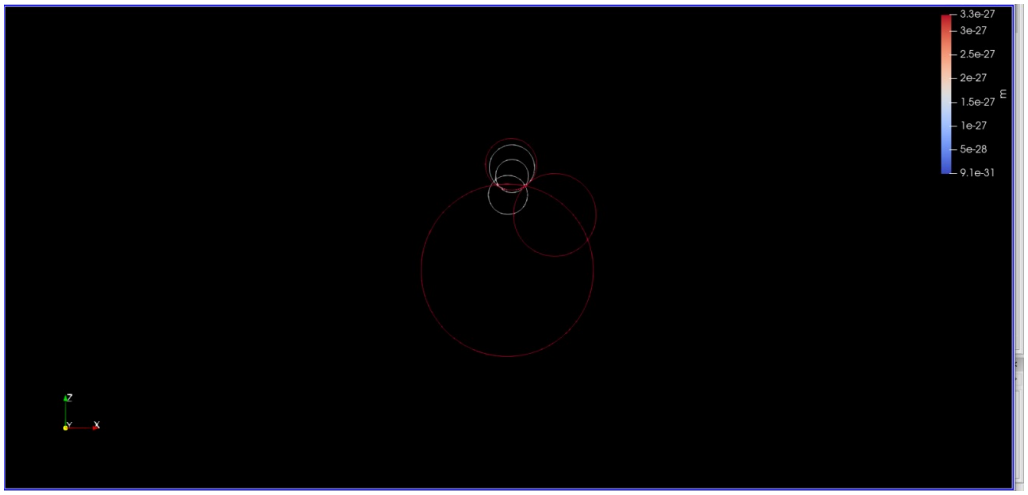


Fig 1.3 Scenario 1, protons and deuterons (front view)

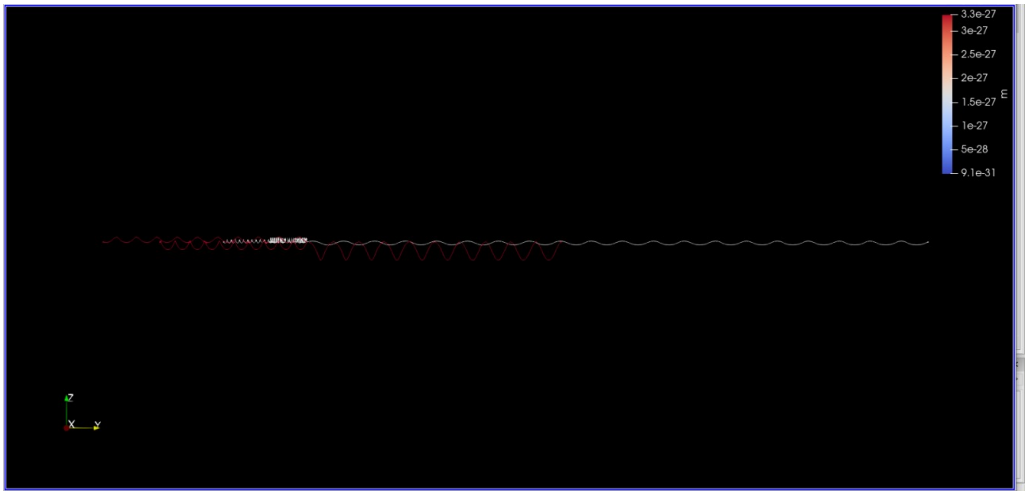


Fig 1.4 Scenario 1, protons and deuterons (side view)

The above figures are of our simulations of particles in uniform magnetic fields, with red, white and blue trails representing protons, deuterons and electrons respectively. The trails serve the purpose of tracing out the path of each particle throughout the simulation. The expected behaviour for charged particles in a uniform magnetic field is to be in circular motion at constant speed, forming circles with constant radius and equally spaced apart. The above simulations do indeed show this behaviour, showing our simulation capable of physically accurately simulating charged particles interacting with a uniform magnetic field.

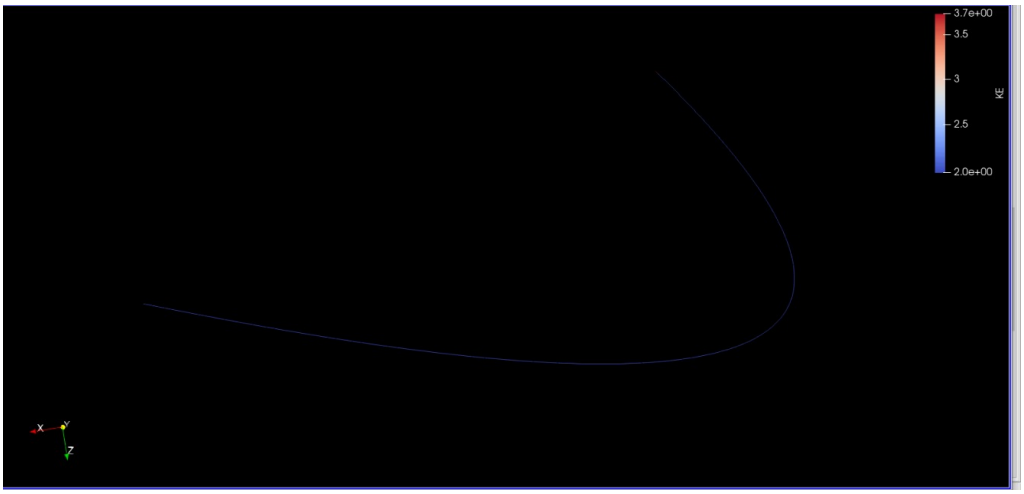


Fig 1.5, Scenario 2

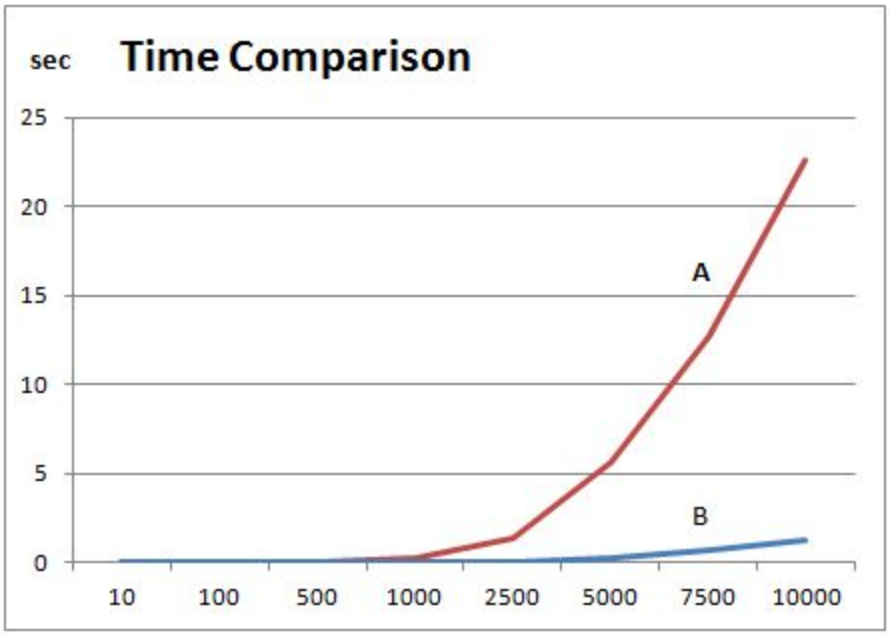
Fig 1.5 shows an electron in a uniform magnetic field, exhibiting parabolic motion as it moves through the uniform electric field. This behaviour is in line with our knowledge on how a charged particle would act when moving through a uniform electric field, namely increasing in speed as well while travelling in the shape of a parabola.

In order to further confirm the legibility and accuracy of our simulation, we can calculate for some points in time where the particle would be, as well as its velocity and acceleration at that point in time, then cross-reference that with the values we obtain in the excel file. This would allow us to be absolutely sure whether our simulation code works accurately and precisely, showing whether or not our solution works.

Future Work

In future research, the simulation code can be further improved by adding code which simulates particle-particle interactions. This can be done through implementing the Biot-Savart Law for magnetic field generated by a point charge to calculate for the magnetic field generated by each particle as well as Coulomb's Law to calculate the electric field generated by each particle, allowing us to accurately simulate the electromagnetic interactions between particles themselves. This would bring more help to nuclear fusion research as nuclear fusion is based on the interactions between particles as well as plasma conditions, so having particle interactions would definitely be a massive benefit in the area of nuclear fusion research.

Optimisation



With the proposed system of optimization, we can compare the runtimes of code using OpenMP or OpenCL (B) with code without any optimization tools. The x-axis represents the input size, which in our context would be *npart,* and the *y-axis* represents the runtime in seconds.

Conclusion

As the project in question is an SMP project, it is undoubtedly a work in progress, with many foreseeable improvements that could be made in the future. The trilinear interpolation could be improved upon, and the types and variety of particle motion could be increased, allowing the optimized code to perhaps be implemented in other fields of scientific research, such as high-energy particle physics or the broader spectrum of particle physics in general. Some possible routes of the foreseeable future would be the implementation of other optimization techniques such as common-time propagation, to allow the further development of the code. Whatever the case, as technology progresses and as our knowledge regarding the subject matter increases, the project can definitely be improved upon.

We would like to thank Professor Paul Lee (NIE-NTU), our external mentor, as well as Mr Sim Mong Chea, our project mentor for helping us with this project.

References

1. Bergman, M. (2019, April 22). *Harvard, Princeton scientists make AI breakthrough for fusion energy*. Harvard Gazette. https://news.harvard.edu/gazette/story/2019/04/harvard-princeton-scientists-make-ai-breakthrough-for-fusion-energy/
2. *Capturing the energy*. (n.d.). ITER. Retrieved February 27, 2021, from https://www.iter.org/sci/MakingitWork#:%7E:text=At%20the%20core%20of%20fusion,matter%20similar%20to%20a%20gas.
3. *Deuterium-tritium fusion*. (n.d.). [Illustration]. Nuclear Fusion. https://en.wikipedia.org/wiki/Nuclear_fusion#/media/File:Deuterium-tritium_fusion.svg
4. *Fission and Fusion: What is the Difference?* (n.d.). Energy.Gov. Retrieved March 7, 2021, from https://www.energy.gov/ne/articles/fission-and-fusion-what-difference
5. Fu, Y., Eldon, D., Erickson, K., Kleijwegt, K., Lupin-Jimenez, L., Boyer, M. D., Eidietis, N., Barbour, N., Izacard, O., & Kolemen, E. (2020). Machine learning control for disruption and tearing mode avoidance. *Physics of Plasmas*, *27*(2), 022501. https://doi.org/10.1063/1.5125581

6. Jeong, S. H., Kim, I. Y., & Hwang, C. K. (2003, March). *Design of a heterodyne electron cyclotron emission system on KSTAR*. Review of Scientific Instruments. https://doi.org/10.1063/1.1530382
7. Kates-Harbeck, J., Svyatkovskiy, A. & Tang, W. Predicting disruptive instabilities in controlled fusion plasmas through deep learning. *Nature* **568,** 526–531 (2019). https://doi.org/10.1038/s41586-019-1116-4
8. Michel, F. (n.d.). *Numerical integration of trajectories in static magnetic fields*. OpenStax CNX. Retrieved February 27, 2021, from https://cnx.org/contents/RELq0Cmx@2/Numerical-integration-of-trajectories-in-static-magnetic-fields.
9. Parfit, M. (2021, February 10). *Future Power: Where Will the World Get Its Next Energy Fix?* National Geographic. https://www.nationalgeographic.com/environment/article/powering-the-future
10. Schussman, G., General Atomics, University of California, Davis, Ma, K. L., Schissel, D., & Evans, T. (2000, October). *Visualizing DIII-D Tokamak Magnetic Field Lines*. https://escholarship.org/content/qt9s4769qq/qt9s4769qq_noSplash_a1e1167431f9ecd8d75036287765ea07.pdf
11. *TEXTOR*. (n.d.). TEXTOR: Tokamak Experiment for Technology Oriented Research. Retrieved March 6, 2021, from https://cer.ucsd.edu/research/fusion-energy/_pages/TEXTOR.html.

12. Wiesemann, K. (2014). A Short Introduction to Plasma Physics. CAS-CERN Accelerator School: Ion Sources - Proceedings. 10.5170/CERN--2013--007.85.