PROJECT REPORT

PROJECT TITLE – SIMULATION OF MOLECULAR DYNAMICS USING OPENMP

COURSE - PARALLEL AND DISTRIBUTED COMPUTING (CSE4001)

SLOT – E2

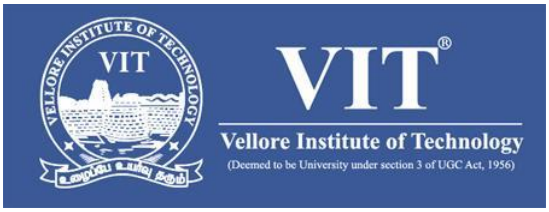
TEAM MEMBER'S DETAILS

NAME	REGISTRATION NUMBER
ZARA IQBAL	19BCE1710
KESHAV BHATIA	19BCE1708
ISAAC EMMANUEL THOMAS	19BCE1750

FACULTY SUPERVISOR

PROF. GAYATHRI R

SCOPE



NOVEMBER, 2021

Table of Content

Abstract:

Molecular-dynamics (MD) simulation is a well-progressed numerical method that shows the use of a suitable algorithm to solve the classical equations of motion for atoms relating with a known inter atomic potential. MD simulation techniques are also well suited for studying surface phenomena, as they give a qualitative explanation of surface structure and dynamics. We are going to parallelize the serial simulation of Molecular Dynamics which will help us to save time.

Key Words: - MD Simulation, Parallel Processing, Computational Modeling, Surface Dynamics.

Chapter -1

Introduction

Molecular Dynamics has been used for many decades now to understand the temperature and pressure dependencies of dynamical concept in liquids, solids, and liquid-solid interfaces. MD simulation techniques are also well matched for understanding surface phenomena, as they give a qualitative understanding of surface structure and dynamics. This Project uses MD methods to better comprehend surface disorder and pre melting.

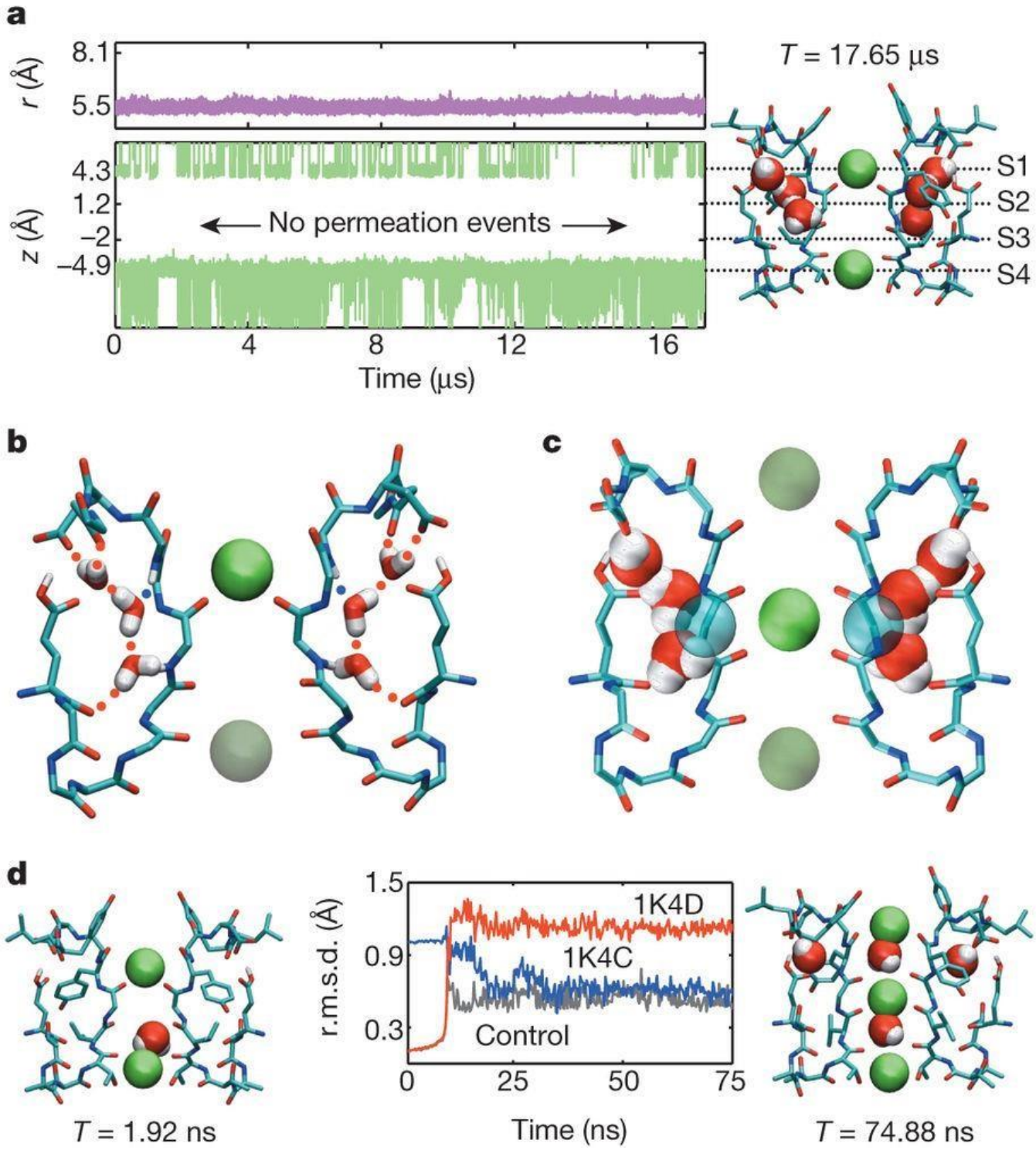
Generally, it examines the temperature dependence of structure and vibration dynamics at surfaces of facecentered cubic (FCC) metals-mostly Ag, Cu, and Ni. It also makes exchange with results from other theoretical and experimental approaches. While the importance in this Project is on metal surfaces, the MD technique has been applied to a wide variety of surfaces with those of semiconductors, insulators, alloys, glasses, and simple or binary liquids. A full review of the pros and cons of the technique as verified to these very exciting systems is ahead the range of this Project.

However, it is important pointing out that the success of the classical MD simulation depends on the accuracy with which the forces acting on the ion cores can be determined. On semiconductor and insulator surfaces, the success of molecular dynamics simulations has made them more appropriate for such designs rather than standard MD simulations.

We can apply MD Simulation on thousands of atoms with computer and we can find out the structure of an atom built on its contact with other atom. This paper tells us about GROMACS one of the applications of MD particularly designed for proteins the effect of this research lead us to a fact that each protein has different long time. From Gromacs we can understand foldings and unfoldings of proteins.

Huge scale molecular dynamics applications and keeping their success in mind we can see large scale simulation playing an always increasing role in future. Mutual simulation and experiments are helping to bond the gap between atomic level properties and whole cell behavior, an endeavor which cannot be done by either of the tactic alone.

MD simulation can deliver the awareness of protein ligand interaction. It demonstrates that free binding energy of membrane proteins should be computed in its natural environment. Binding constants give information of atomistic properties and functionality of proteins and at the same time are also helpful for Drug transposition.



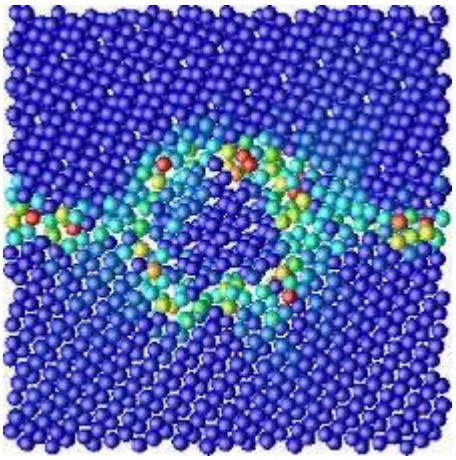
Key Properties of MD

1. **Atoms never stop jiggling**

In real life, and in an MD simulation, atoms are in constant motion. They will not proceed to an energy minimum and keep there.

Given enough time, the simulation examples the Boltzmann distribution

- That is, the probability of detecting a specific arrangement of atoms is a function of the potential energy.
- In reality, one often does not simulate long enough to reach all energetically favorable arrangements.
- This is not the only way to explore the energy surface (i.e., sample the Boltzmann distribution), but it's a pretty effective way to do so.



2. **Energy Conservation**

Total energy (potential + kinetic) should be conserved

- In atomic arrangements with lower potential energy, atoms move faster.
- In practice, total energy inclines to grow gradually with time due to numerical errors (passing errors).
- In a lot of simulations, one adds a mechanism to keep the temperature unevenly constant (a "thermostat")

3. **Water is Important**

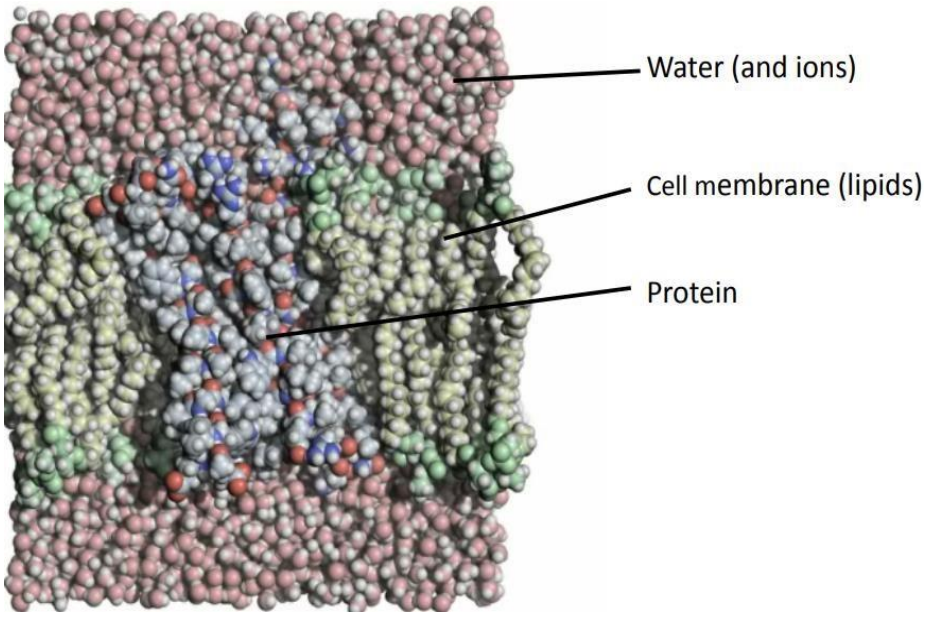
Overlooking the solvent (the molecules nearby the molecule of interest) leads to main artifacts examples are Water, salt ions (e.g., sodium, chloride), lipids of the cell membrane.

Two choices for taking solvent into account – Plainly denote solvent molecules

- High computational cost but more precise.

- Usually accept periodic boundary settings (a water molecule that drives off the left side of the simulation box will come back in the right side, like in PacMan).
- Implicit solvent
- Mathematical model to imprecise average effects of solvent. Less precise but faster

4. **Explicit Solvent**



Limitation of MD Simulation

1. **Time Scale**

Simulations need short time stages for numerical stability. 1 time step ≈ 2 fs (2×10–15 s).

Structural changes in proteins can take nanoseconds (10–9 s), microseconds (10–6 s), milliseconds (10–3 s), or longer – Millions to trillions of sequential time steps for nanosecond to millisecond events (and even more for slower ones)

Until lately, simulations of 1 microsecond were unusual. Advances in computer power have allowed microsecond simulations, but simulation periods continue a challenge.

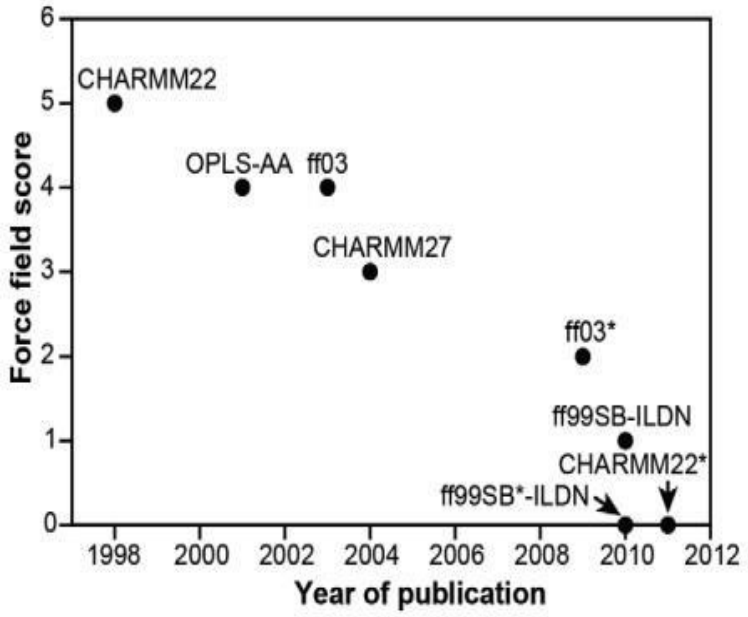
Enabling longer-timescale simulations is an active research area, connecting:

- Algorithmic developments
- Parallel computing
- Hardware: GPUs, specialized hardware

2. Force Field Accuracy

Molecular mechanics force fields are integrally calculations. They have improved substantially over the last 10 years, but many boundaries still remains.

Below force fields with lower scores are better, as measured by agreement between simulations and experimental data. Even the force fields with scores of zero are imperfect, however!



In repetition, one needs some experience to know what to trust in a simulation.

3. Covalent bonds cannot break or form during (standard) MD simulations

Once a protein is formed, most of its covalent bonds don't break or form during characteristic function.

A few covalent bonds form and break more normally (in real life): –

- Disulfide bonds between cysteines.
- Acidic or basic amino acid residues can lose or gain a hydrogen (i.e., a proton)

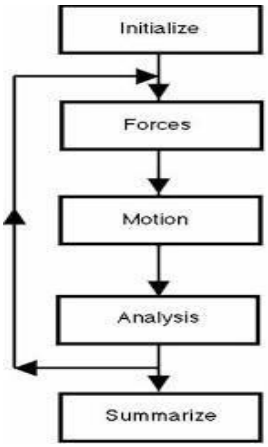
Proposed Work

We will write a code which will expect the kinetic and potential energy of an atom at all time step which will help us in foreseeing the motions (velocity, position acceleration etc) of the atoms. The sum of kinetic energy and potential energy must be constant and equal to total energy our code will also print the error in total energy which will tell us about what percentage of error we can get in our calculations.

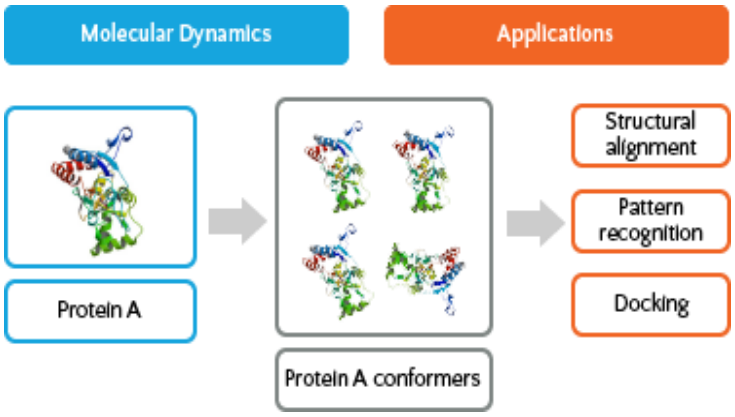
Below we have given the problem analysis chart (PAC) of our simulation code for better accepting of the MD Simulation.

Input	Processing	Output	Solution Alternative
Spatial Dimension No. of particles in Simulation No. of Time Steps Size of Each Time step should also print the relative error in the Execution.	At every step, our simulation should the potential and kinetic energies. The sum of these energies should be a constant. As an accuracy check, our simulation total energy. We should parallelize the code and associate the time taken in both serial and parallel execution.	Relative Error in Total energy. Time difference between Serial and parallel	NONE report

The main outcome we are expecting from this project is the decrease in amount of time taken to compute the potential and kinetic energies of atoms and that is also the reason why we are paralleling the compute and update function as they do most of the calculation. Another Outcome which we are expecting from this simulation is decrease in the error of theoretical and practically observed energies. Lesser the error, more the accuracy but we also know we cannot have the error free output but we are doing our best to make it very small by do not taking approximation of most of the values in calculation part.



Application of molecular dynamics are shown in below figure



Our application code will have the following functions

- **Compute**: - COMPUTE computes the forces and energies of atoms.
- **Initialize**: - INITIALIZE initializes the positions, velocities, and accelerations.
- **Timestamp**: - TIMESTAMP prints the current YMDHMS date as a time stamp.
- **Update**: - UPDATE updates positions, velocities and accelerations.

➢ **Displacement**: - DISPLACEMENT computes the displacement between two particles.

- **CPU Time**: - CPU_TIME reports the elapsed CPU time.

Compute and update function are the most time overriding functions as they are only about computation and solving mathematical equations so we are going to parallelyze both of them. Some parallel function which we are about to use in our code are **pragma omp parallel, pragma omp reduction, pragma omp for, pragma omp shared, pragma omp private.** The major extensive thing in our project will be the time difference in the serial code and parallel code because the main goal of our project is to decrease the time taken in molecular dynamics simulation.

In mathematics, numerical analysis, and numerical partial differential equations, domain decay methods solve a edge value problem by splitting it into smaller boundary value problems on sub domains and repeating to coordinate the solution between adjacent sub domains. A coarse problem with one or few unknowns per sub domain is used to further organize the solution between the sub domains globally. The problems on the sub domains are independent, which makes domain decay methods suitable for parallel calculating.

Chapter 3

Calculation Involved

The Basic algorithm is we divide time into discrete time steps, no more than a few femtoseconds (10–15 s) each and at each time step we calculate the forces performing on each atom, using a molecular mechanics force field and when the drive in atom occurs we update position and velocity of each atom using Newton's laws of motion.

We have simple equations of motion and used them to do the computations.

Newton's second law: F = ma – where F is force on an atom, m is mass of the atom, and a is the atom's acceleration Recall that: – where x represents coordinates of all atoms, and U is the potential energy function

Velocity is the derivative of position, and acceleration is the derivative of velocity. We can thus write the equations of motion as: $F(x) = -\nabla U(x)$

$$dx/dt = v \quad \text{and} \quad dv/dt = F(x)/m$$

This is a system of ordinary differential equations – For n atoms, we have 3n position coordinates and 3n velocity coordinates. "Analytical" (algebraic) solution is difficult but Numerical solution is straightforward.

$$x_{i+1} = x_i + \delta_t v_i \quad v_{i+1}$$

$$= v_i + \delta_t F(x_i)/m$$

Above is the numerical open solution where δ_t is the time step.

In practice, people use “time symmetric” integration methods such as “Leapfrog Verlet”

$$x_{i+1} = x_i + \delta_t \, v_{(i+1)/2}$$

$$v_{(i+1)/2} = v_{(i-1)/2} + \delta_t \, F(x_i) / m$$

This gives more correctness.

Chapter 4

Accelerating MD Simulation

As we know MD simulation takes a lot of time so we need to speed up and to do that first we need to find out the reasons why it takes so much time then we have to provide solution for these problems.

Reasons for computationally intense MD

• Many time steps (millions to trillions).

• Substantial amount of computation at every time step

  ➢ Controlled by non-bonded interactions, as these implement between every pair of atoms.

• In a system of N atoms, the number of non-bonded terms is proportional to N2

  ➢ Can we overlook interactions beyond atoms divided by more than some fixed cutoff distance?

    • For van der Waals interactions, yes. These forces fall off quickly with distance.

    • For electrostatics, no. These forces fall off slowly with distance.

Methods to speed up MD Simulation

• Reduce the amount of calculation per time step

• Reduce the number of time steps needed to simulate a certain amount of physical time

• Reduce the amount of physical time that must be simulated

• Parallelize the simulation across multiple computers

• Redesign computer chips to make this computation run faster

Reduce the amount of computation per time step -> Faster algorithms. Example: fast estimated methods to compute electrostatic interactions, or methods that allow you to assess some force field terms every other time step.

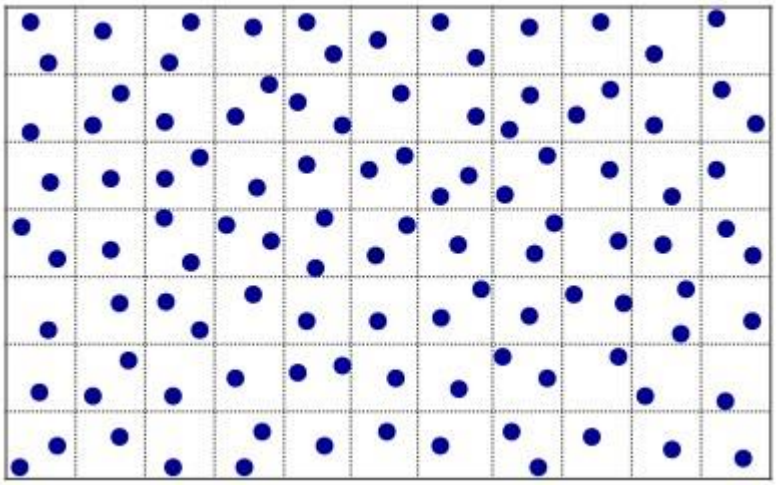
Reduce the number of time steps required to simulate a certain amount of physical time -> One can increase the time step several fold by slowing out some very fast motions (e.g., certain bond lengths).

Reduce the amount of physical time that must be simulated -> A major research area includes making events of interest take place more quickly in simulation, or making the simulation reach all low-energy conformational states more quickly.

For example, one might implement artificial forces to pull a drug molecule off a protein, or push the simulation away from states it has already stayed. Each of these methods is active in certain specific cases.

Parallelize the simulation across multiple computers -> Splitting the computation associated with a single time step across multiple processors needs communication between processors. In our project we are about to use this method.

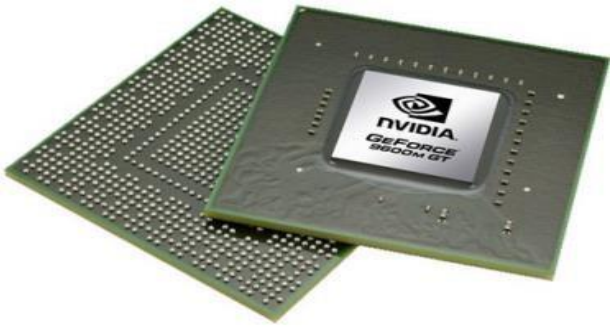
- Generally each processor takes accountability for atoms in one spatial region.
- Algorithmic improvements can reduce communication necessities.



Alternative approach -> perform many short simulations. One research goal is to use short simulations to predict what would have happened in a longer simulation.

Redesign computer chips to make this computation run faster

- GPUs (graphics processor units) are now extensively used for MD simulations. They pack more arithmetic logic on a chip than traditional CPUs, and give a substantial speedup.
  - Parallelizing through multiple GPUs is difficult.
- Several projects have built chips especially for MD simulation.
  - These pack even more arithmetic logic onto a chip, and allow for parallelization through many chips.



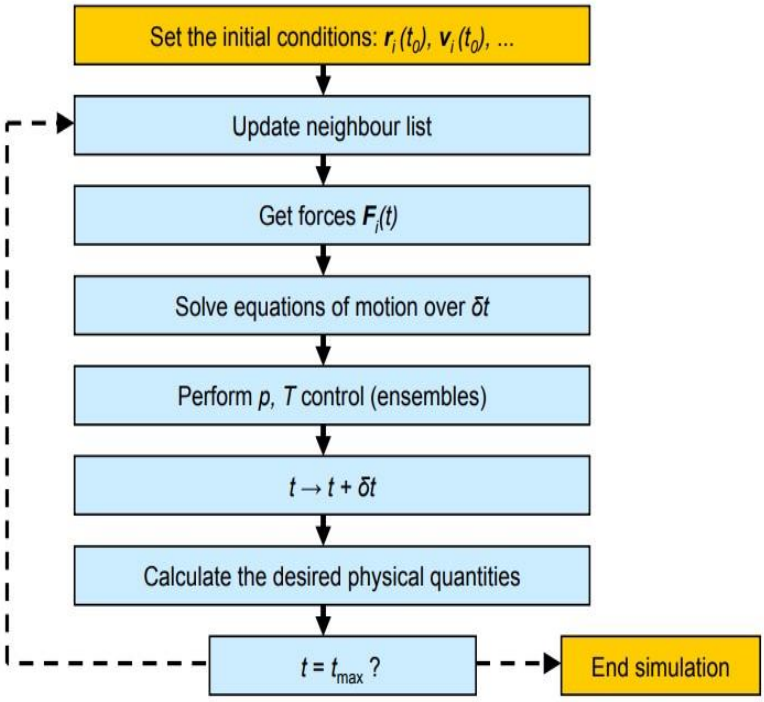
GPU

Specialized chip

Chapter 5

Implementation

Below given is the implementation chart which our algorithm is about to follow to get our wanted result.



Boundary Conditions

Our spatial scales are narrow. What can we do? ƒ

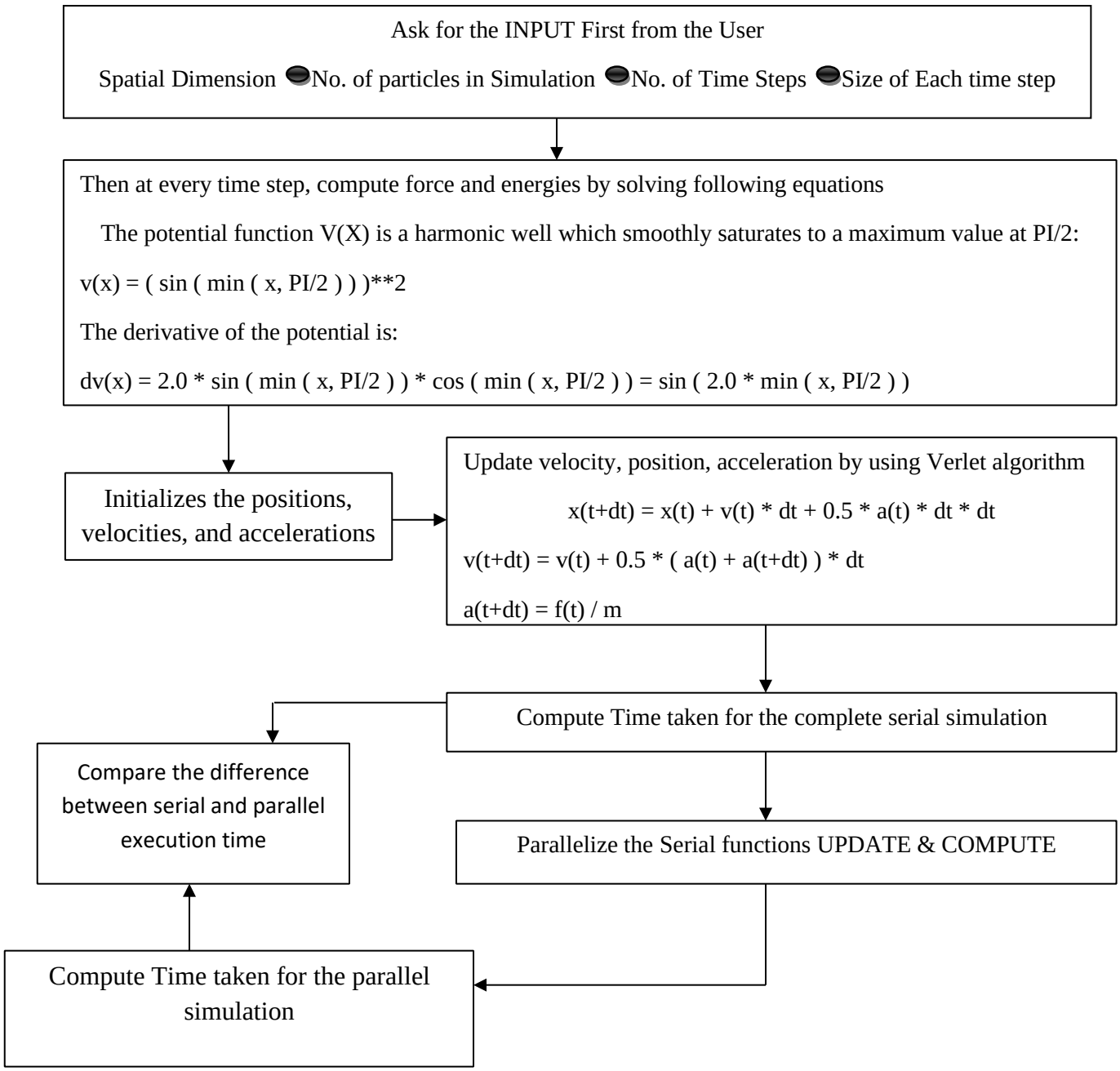
- Periodic boundaries the most widely used option ƒ
- Simulations on a sphere ƒ
- Mixed boundaries for example, infinite in z, periodic in x and y ƒ
- No boundaries e.g. droplet of liquid, protein in vacuum; involves a formation of a surface on the system ƒ
- Fixed limits totally unphysical, should be evaded; in principle we need a 'sacrificial region' between the part of the system under study and a fixed limit.

Periodic Boundary Conditions

- Particles crossing a boundary of the simulation cell appear back on the opposite side.
- Basically we try to model an infinite (~macroscopic) volume of matter with a small, finite simulation cell.
- The minimum image convention sets boundaries on the interaction cut-off length used in the simulations.

Least image convention: - each particle interacts only once with a given particle; by the particle proper or its periodic image, which one ever is closer.

Flow Chart of our Implementation is as follows.



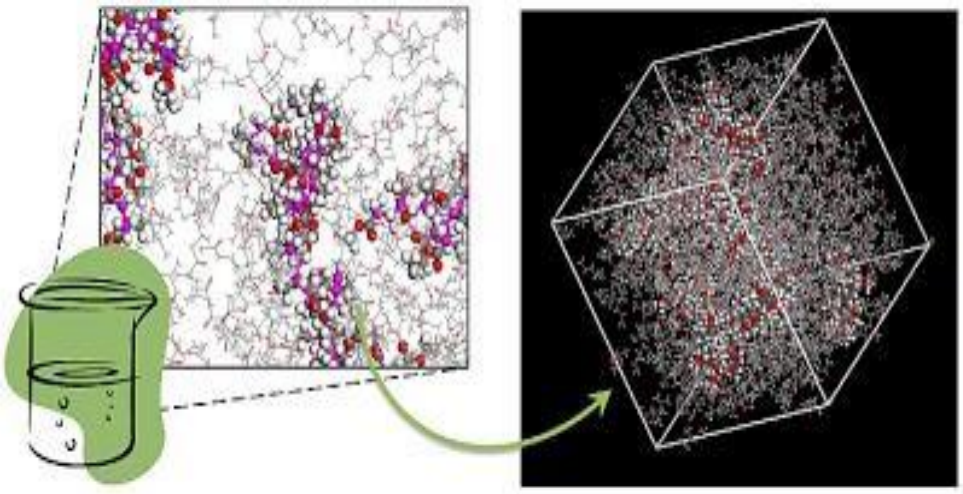
Calculating the interactions As a result of using periodic limit conditions, each particle is in the simulation box appears to be relating not only with the other particles in the box, but also with their images. Actually, the number of interacting pairs increases extremely. Nevertheless, this problem can be overcome if one uses a potential with a finite range, since the communication of two particles separated by a distance exceeding a cut off distance can be overlooked. Assuming that the box size is larger than 2Rcut along each Cartesian direction, it is clear that at most one among the pairs made by a particle i in the box and all the periodic images of another particle j will lie inside the cut off distance Rcut and, thus, will interact. This is the kernel of the minimum image principle: among all images of a particle, study only the closest and neglect the rest.

Correcting particle coordinates After each integration step the coordinates of the particles must be studied. If a particle has left the simulation region, its coordinates must be readjusted to bring it back inside, which is conforming to bring in an image particle through the opposite boundary. Supposing that the simulation region is a rectangular box, this is done by adding to or subtracting from the affected particle coordinate the size L of the box along the conforming direction. Thus, for instance, supposing that the xcoordinates are defined to lie between 0 and L, if $x < 0$ (the particle leaves the box in the negative Ox direction), then the coordinate must be corrected by adding the box size L. If $x>L$ ˙ (the particle exits in the positive Ox direction), L must be subtracted from the particle coordinate:

$$\text{if } x < 0 \text{ then } x \rightarrow x + L$$

$$\text{if } x>L \text{ ˙ then } x \rightarrow x - L.$$

An alike analysis must be carried out for each coordinate of the particle. The readjustment of the coordinates in the context of using periodic limit conditions has been applied in the following routine.



Chapter 6

Code:-

```
#include<stdlib.h>

#include<stdio.h>

#include<time.h>

#include<math.h> #include<omp.h>
```

```
int main ( int argc, char *argv[] ); void compute ( int np, int nd,
double pos[], double vel[],   double mass, double f[], double *pot,
double *kin ); double dist ( int nd, double r1[], double r2[], double
dr[] ); void initialize ( int np, int nd, double box[], int *seed,
double pos[],   double vel[], double acc[] ); double r8_uniform_01
( int *seed ); void timestamp ( void ); void update ( int np, int nd,
double pos[], double vel[], double f[],   double acc[], double mass,
double dt ); int main ( int argc, char *argv[] )
{   double *acc;
double *box;
double dt = 0.0001;
double e0;   double
*force;
  int i;
int id;
  double kinetic;
double mass = 1.0;
  int nd = 3;
int np = 1000;
double *pos;
double
potential; int
proc_num;   int
seed =
123456789;
  int step;
  int step_num = 400;
  int step_print;   int
step_print_index;   int
step_print_num;
double *vel;   double
```

```
wtime;   timestamp (
);
  proc_num = omp_get_num_procs ( );
  acc = ( double * ) malloc ( nd * np * sizeof ( double ) );
box = ( double * ) malloc ( nd * sizeof ( double ) );   force
= ( double * ) malloc ( nd * np * sizeof ( double ) );   pos
= ( double * ) malloc ( nd * np * sizeof ( double ) );   vel =
( double * ) malloc ( nd * np * sizeof ( double ) );
  printf ( "\n" );
  printf ( "MD_OPENMP\n" );
  printf ( "  C/OpenMP version\n" );
  printf ( "\n" );
  printf ( "  A molecular dynamics program.\n" );
  printf ( "\n" );
  printf ( "  NP, the number of particles in the simulation is %d\n", np );
printf ( "  STEP_NUM, the number of time steps, is %d\n", step_num );
  printf ( "  DT, the size of each time step, is %f\n", dt );
printf ( "\n" );
  printf ( "  Number of processors available = %d\n", omp_get_num_procs ( ) );
printf ( "  Number of threads =              %d\n", omp_get_max_threads ( ) ); /*
  Set the dimensions of the box.
*/
  for ( i = 0; i < nd; i++ )
  {     box[i] =
10.0;
  }   printf ( "\n" );   printf ( "  Initializing positions, velocities, and
accelerations.\n" );
/*
  Set initial positions, velocities, and accelerations.
*/   initialize ( np, nd, box, &seed, pos, vel,
acc );
```

```
/*
  Compute the forces and energies.
*/   printf ( "\n" );   printf ( "  Computing initial forces and
energies.\n" );  compute ( np, nd, pos, vel, mass, force,
&potential, &kinetic );   e0 = potential + kinetic;
/*
  This is the main time stepping loop:
    Compute forces and energies,
    Update positions, velocities, accelerations.
*/   printf ( "\n" );   printf ( "  At each step, we report the potential and kinetic
energies.\n" );   printf ( "  The sum of these energies should be a constant.\n"
);    printf ( "  As an accuracy check, we also print the relative error\n" );
printf ( "  in the total energy.\n" );    printf ( "\n" );    printf ( "       Step
Potential       Kinetic        (P+K-E0)/E0\n" );   printf ( "                Energy P
Energy K       Relative Energy Error\n" );   printf ( "\n" );

  step_print = 0; step_print_index
= 0;   step_print_num = 10;
  step = 0;
  printf ( "  %8d  %14f  %14f  %14e\n",
    step, potential, kinetic, ( potential + kinetic - e0 ) / e0 );
  step_print_index = step_print_index + 1;   step_print = (
step_print_index * step_num ) / step_print_num;   wtime =
omp_get_wtime ( );   for ( step = 1; step <= step_num; step++ )
  {
    compute ( np, nd, pos, vel, mass, force, &potential, &kinetic );
if ( step == step_print )
    {
      printf ( "  %8d  %14f  %14f  %14e\n", step, potential, kinetic,
       ( potential + kinetic - e0 ) / e0 );
```

```
      step_print_index = step_print_index + 1;       step_print = (
step_print_index * step_num ) / step_print_num;
    }
    update ( np, nd, pos, vel, force, acc, mass, dt );
  }
  wtime = omp_get_wtime ( ) - wtime;
  printf ( "\n" );
  printf ( "  Elapsed time for main computation:\n" );
printf ( "  %f seconds.\n", wtime );
  free ( acc );
free ( box );
free ( force );
free ( pos );
free ( vel );
/*
  Terminate.
*/

  printf ( " end of execution.\n" );   printf ( "\n" );
timestamp ( );   return 0; } void compute ( int np, int nd,
double pos[], double vel[],   double mass, double f[],
double *pot, double *kin )
{   double
d;   double
d2;
  int i;
int j;   int
k;
  double ke;   double pe;   double PI2 =
3.141592653589793 / 2.0;
```

```
 double rij[3];
pe = 0.0;   ke
= 0.0;
# pragma omp parallel \
 shared ( f, nd, np, pos, vel ) \
private ( i, j, k, rij, d, d2 )
# pragma omp for reduction ( + : pe, ke )
for ( k = 0; k < np; k++ )
 {
/*
 Compute the potential energy and forces.
*/    for ( i = 0; i < nd;
i++ )
   {
     f[i+k*nd] = 0.0;
   }

   for ( j = 0; j < np; j++ )
   {      if ( k
!= j )
     {
       d = dist ( nd, pos+k*nd, pos+j*nd, rij );
/*
 Attribute half of the potential energy to particle J.
*/        if ( d <
PI2 )        {
d2 = d;
       }
       else
{
         d2 = PI2;
```

```
        }
        pe = pe + 0.5 * (pow( sin(d2),2));
for ( i = 0; i < nd; i++ )
        {
          f[i+k*nd] = f[i+k*nd] - rij[i] * sin( 2.0 * d2 ) / d;
        }
      }
    }
/*
  Compute the kinetic energy.
*/     for ( i = 0; i < nd;
i++ )
    {
      ke = ke + vel[i+k*nd] * vel[i+k*nd];
    }
  }
  ke = ke * 0.5 * mass;
  *pot = pe;
*kin = ke;
return;
}
/******************************************************************************/
double dist ( int nd, double r1[], double r2[], double dr[] )
/******************************************************************************/
{   double
d;
  int i;
  d = 0.0;    for ( i = 0; i
< nd; i++ )
```

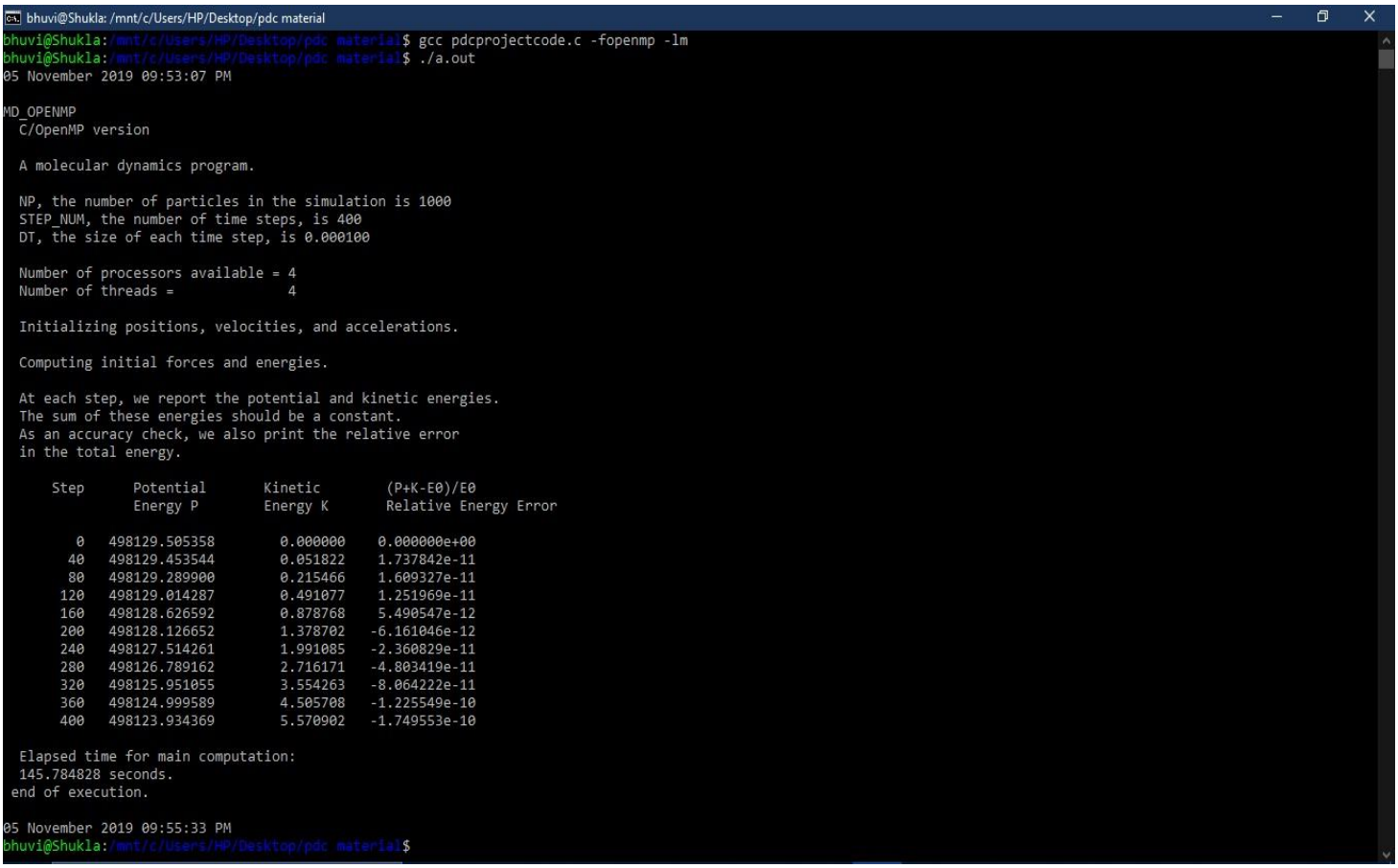
```
  {     dr[i] = r1[i] -
r2[i];     d = d + dr[i] *
dr[i];
  }   d =
sqrt(d);
return d;
}
/******************************************************************************/
void initialize ( int np, int nd, double box[], int *seed, double pos[],
double vel[], double acc[] )
/******************************************************************************/
{
  int i;
int j;
/*
  Give the particles random positions within the box.
*/   for ( i = 0; i < nd;
i++ )
  {     for ( j = 0; j < np;
j++ )
    {
      pos[i+j*nd] = box[i] * r8_uniform_01 ( seed );
    }   }   for ( j = 0; j <
np; j++ )
  {     for ( i = 0; i < nd;
i++ )
    {
      vel[i+j*nd] = 0.0;
    }   }   for ( j = 0; j <
np; j++ )
```

```
  {     for ( i = 0; i < nd;
i++ )
    {
      acc[i+j*nd] = 0.0;
    }   }
return;
}
/******************************************************************************/
double r8_uniform_01 ( int *seed )
/******************************************************************************/
{
  int k;
  double r;   k =
*seed / 127773;
*seed = 16807 * (
*seed - k * 127773 )
- k * 2836;

  if ( *seed < 0 )
  {
    *seed = *seed + 2147483647;
  }
  r = ( double ) ( *seed ) * 4.656612875E-10;
  return r;
}
/******************************************************************************/
void timestamp ( void )
/******************************************************************************/
{
```

```
# define TIME_SIZE 40   static char
time_buffer[TIME_SIZE];   const
struct tm *tm;   size_t len;
  time_t now;   now = time ( NULL );   tm = localtime ( &now );   len = strftime
( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm );
  printf ( "%s\n", time_buffer );
return;
# undef TIME_SIZE
}
/******************************************************************************/
void update ( int np, int nd, double pos[], double vel[], double f[],
double acc[], double mass, double dt )
/******************************************************************************/
{
  int i;
int j;
  double rmass;

  rmass = 1.0 / mass; #
pragma omp parallel \
  shared ( acc, dt, f, nd, np, pos, rmass, vel ) \
  private ( i, j )
# pragma omp for
  for ( j = 0; j < np; j++ )
  {     for ( i = 0; i < nd;
i++ )
    {
      pos[i+j*nd] = pos[i+j*nd] + vel[i+j*nd] * dt + 0.5 * acc[i+j*nd] * dt * dt;
vel[i+j*nd] = vel[i+j*nd] + 0.5 * dt * ( f[i+j*nd] * rmass + acc[i+j*nd] );       acc[i+j*nd]
= f[i+j*nd] * rmass;
```

```
    }   }
return;
}
```

Screenshot of Output:-



Conclusion

Molecular dynamics is an significant computational tool to simulate and comprehend biochemical processes at the atomic level. However, precise simulation of processes such as protein folding needs a large number of both atoms and time steps. This in turn leads to huge runtime requirements. Hence, looking for fast solutions is of highest importance to research.

In this project we present a new approach to accelerate molecular dynamics simulations with inexpensive product graphics hardware. To derive an effective mapping onto this type of computer architecture, we have used the new Compute Unified Device Architecture programming interface to implement a new parallel algorithm.

Our experimental results show that the parallel algorithm based method allows speedups of up to fifteen seconds compared to the corresponding sequential implementation. This speed up can also be enhanced when we increase number of atoms and number of time steps.

The energy or force calculation is the most time using part of almost all Molecular dynamics Simulation. If we take a model system with pair wise additive interaction (as done in many molecular simulation), we have consider the contribution to the forces on the particle I by all it neighbor. If we do not shorten the interaction, this implies that, for a system of N particles, we must evaluate N(N-1)/2 pair interaction.

Even if we do shorten the potential, we still would have to compute all N(N-1)/2 pair distances to define which pairs can interact. This implies that, if we use no tricks, the time needed for assessment of the energy scales as N^2. There exit efficient techniques for speeding up the assessment of both short-term and longterm contact in such a way that the computing time scales as $N^{3/2}$, rather than N^2. The technique which we have used is combination of Verlet and cell lists.

The first question that rises is when to use which method. This depends mostly on the details of the system. In any event, we always start with a arrangement as simple as possible, hence no trick at all. Although the algorithm scales as N^2, it is straightforward to implement and thus the probability of programming errors id relatively small. In addition we should consider how often the program will be used.

The use of verlet list become beneficial if the number of particles in the list is significantly less than the total number of particles in three dimension this means

$$n_v = (4/3)*\pi r^3 \rho << N$$

After completing our project finally we can accomplish that writing code in OpenMP for molecular dynamics simulation helped in decreasing time of evaluation of forces or energies which was our main purpose when we choose this topic as our project and we are happy that we have successfully reduced time period by 15 seconds.

Future Research

We can include Graphics processing units (GPUs), originally developed for interpreting real-time effects in computer games, now provide unparalleled computational power for scientific applications. We can develop a general purpose molecular dynamics code that runs completely on a single GPU. It is expected that our GPU application can provide a presentation equivalent to that of fast 30 processor core distributed memory cluster. Results of this can show that GPUs implementation as an inexpensive another to such clusters and discuss implications for the future.

Currently, the peak presentation of state-of-the-art GPUs is roughly ten times faster than that of comparable CPUs. Furthermore, the growth rate of the number of transistors used on GPUs is greater than for microprocessors. Therefore, GPUs will become an even more useful alternative for high performance calculating in the near future. And these facts aim and excite us to apply molecular dynamics using GPU.

The *Compute Unified Device Architecture* (CUDA) [2] is a new hardware and software architecture for delivering and managing computations on GPUs. It treats the GPU as a data-parallel computing device without the need of mapping calculations to the graphics pipeline. By using the standard C language, CUDA makes it easy GPU-based software development for scientific calculating compared to previously used graphics-oriented languages such as OpenGL or Cg.

Molecular dynamics (MD) is a computationally concentrated method of studying the natural time-evolution of a system of atoms using Newton's classical equations of motion. In practice, MD has always been limited more by the current available computing power than by investigators' resourcefulness. Researchers in this field have typically concentrated their efforts on simplifying models and finding what may be neglected while still obtaining acceptable results.

MD simulations can benefit from the calculating power of GPUs. In order to exploit the GPU's capabilities for high performance MD simulation, we present a new algorithm for non-bonded short-range connections within the atom system. The algorithm has been applied using C++ and CUDA and tested on a physical system of up to 131,072 atoms. We show that our new MD algorithm leads to a performance development of one order of scale on a product NVIDIA GeForce 8800 GTX card.

References

1. S. Alam, P. Agarwal, M. Smith, J. Vetter, D. Caliga "**Using FPGA devices to accelerate biomolecular simulations"** Computer, 40 (3) (2007), pp. 66-73

2. M.P. Allen, Introduction to molecular dynamics simulation, in: Computational Soft Matter—From Synthetic Polymers to Proteins, NIC Series, vol. 23, John von Neumann Institute for Computing, 2004, pp. 1–28

3. A.G. Anderson, W.A. Goddard, P. Schroder **Quantum Monte Carlo on graphical processing units** Computer Physics Communications, 177 (2007), pp. 298-306

4. J.A. Anderson, C.D. Lorenz, A. Travesset **General purpose molecular dynamics simulations fully implemented on graphics processing units,** Journal of Computational Physics, 227 (2008), pp. 5342-5359

5. R.G. Belleman, J. Bédorf, S.F. Portegies Zwart **High performance direct gravitational *N*-body simulations on graphics processing units II: An implementation in CUDA,** New Astronomy, 13 (2008), pp. 103-112

7. Mangiardi C.M. (Master's thesis), Molecular-Dynamics Simulations using Spatial Decomposition and Task-Based Parallelism, Laurentian University, Sudbury, Ontario (2016)

8. Pal A., Agarwala A., Raha S., Bhattacharya B., J. Parallel Distrib. Comput., 74 (2014), pp. 2203-2214

9. Zhiwei Zhang; Pei Chen; Fei Qin, "Molecular dynamics simulation on subsurface damage layer during nano grinding process of silicon wafer", 18th International Conference on Electronic Packaging Technology (ICEPT) pp. 487-490,2017

10. Soni S, Tyagi C, Grover A[1], Goswami SK, "Molecular modeling and molecular dynamics simulations based structural analysis of the SG2NA protein variants" ,NCBI ,2014.

11.M.P. Allen, D. Frenkel, J. Talbot **Molecular dynamics simulation using hard particles** , Comput. Phys. Rep., 9 (1989), pp. 301-353