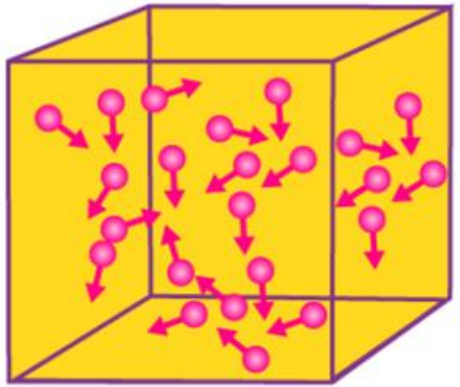
OPENMP END TERM PROJECT

Group H

Project Title: Many Body Collisions

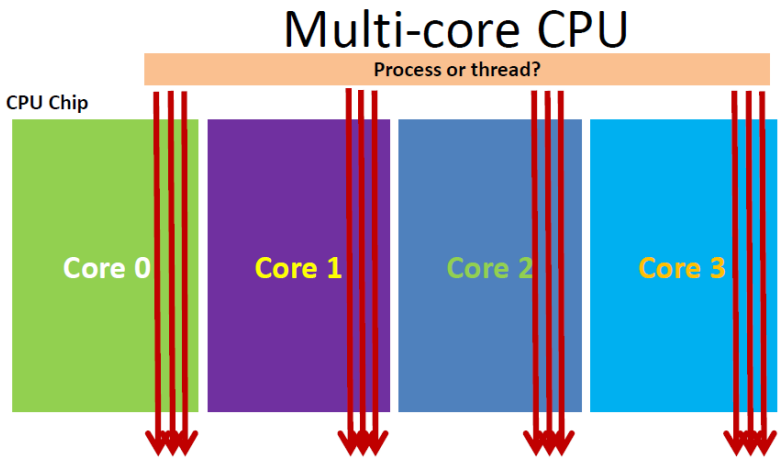
Team members

- *Jothi Prakash - 19EC30028*
- *Rekha Lokesh - 19EC10052*
- *Sampara Sai Charan - 19EC10057*
- *Chaitanya Bhargav N - 19EC10016*
- *Geddam Ashlesh Kumar - 19EC10080*



Open MP

OpenMp allows us to perform parallel execution of code on the machine using the predefined facilities that are defined in it. This allows us to use the full capability of modern systems to increase efficiency and throughput of the systems.



Multi Body Simulations

OPENMP - CPP Code

Reading And Writing To The Files

File Stream is used to read the input from the Trajectory.txt file.

File Stream is used to write the output to the Coordinates.txt file.

```
// File stream to read the input and write the output
fstream fptr;
fstream res;
res.open("Coordinates.txt", ios::out);

fptr.open("Trajectory.txt", ios::in);
```

Storing The Input Values

```
// Getting the variable values
string line;
double width, length, depth;
int body_cnt;
double del_t, mass, radius;

getline(fptr, line);
line = cleanString(line);
width = stod(line);
getline(fptr, line);
line = cleanString(line);
length = stod(line);
getline(fptr, line);
line = cleanString(line);
depth = stod(line);
getline(fptr, line);
line = cleanString(line);
body_cnt = stoi(line);
getline(fptr, line);
line = cleanString(line);
radius = stod(line);
getline(fptr, line);
line = cleanString(line);
mass = stod(line);
getline(fptr, line);
line = cleanString(line);
del_t = stod(line);

getline(fptr, line);
```

Simulation In Every Time Step

```
// Measuring the time
double start;
double end;
start = omp_get_wtime();

// Code for the simulation
for (int n = 0; n < totalTime; n++)
{
```

Simulation Step One

Firstly, we need to calculate the force acting on each particle using every other particle in the environment.

This is done by calculating the distance between them and applying the force formula, to derive the Vector force acting on them.

This part is highly independent and therefore can be parallelized completely.

To make it more efficient we can collapse the 2 loops for better efficiency.

```
            // Finding the force on each body due to interbody forces
#pragma omp parallel for num_threads(threadCnt) collapse(2)
            for (int i = 0; i < body_cnt; i++)
            {
                for (int j = 0; j < body_cnt; j++)
                {
                    if (i == j)
                        continue;
                    double dis = coord[j][0] - coord[i][0];
                    if (dis >= 0.0)
                    {                double dis
                        dis = max(dis, minDis);
                    }
                    else
                    {
                        dis = min(dis, -minDis);
                    }
                    force[i][0] = (mass * mass) / (dis * dis);
                    dis = coord[j][1] - coord[i][1];
                    if (dis >= 0.0)
                    {
                        dis = max(dis, minDis);
                    }
                    else
                    {
                        dis = min(dis, -minDis);
                    }
                    force[i][1] = (mass * mass) / (dis * dis);
                    dis = coord[j][2] - coord[i][2];
                    if (dis >= 0.0)
                    {
                        dis = max(dis, minDis);
```

Simulation Step Two

After calculating the force, we need to calculate the change in velocity of each body due to the force acting on it.

This is done in 2 intermediate half steps, where we first update the velocity by half the required value. Then we calculate the coordinates. Then we again increase the velocity by another half the required value.

This region can also be parallized, as all the velocity and coordinate updates are independent of each other.

```
            // Changing the velocity and coordinates based on the forces
#pragma omp parallel for num_threads(threadCnt)
            for (int i = 0; i < body_cnt; i++)
            {

                velocity[i][0] += del_t_by_m * force[i][0] * 0.5;
                velocity[i][1] += del_t_by_m * force[i][1] * 0.5;
                velocity[i][2] += del_t_by_m * force[i][2] * 0.5;

                coord[i][0] += velocity[i][0] * del_t;
                coord[i][1] += velocity[i][1] * del_t;
                coord[i][2] += velocity[i][2] * del_t;

                velocity[i][0] += del_t_by_m * force[i][0] * 0.5;
                velocity[i][1] += del_t_by_m * force[i][1] * 0.5;
                velocity[i][2] += del_t_by_m * force[i][2] * 0.5;

                if (coord[i][0] > width || coord[i][0] < 0)
                {
                    velocity[i][0] = -velocity[i][0];
                    coord[i][0] = min(coord[i][0], width);
                    coord[i][0] = max(coord[i][0], 0.0);
                }

                if (coord[i][1] > length || coord[i][1] < 0)
                {
                    velocity[i][1] = -velocity[i][1];
                    coord[i][1] = min(coord[i][1], length);
                    coord[i][1] = max(coord[i][1], 0.0);
                }
```

Simulation Step Three

Once we have updated the velocity and the coordinates, we need to check about the collision between the particles.

This can be done by measuring the distance between the 2 particles and applying Newton's Law of Momentum Conservation as the collision is assumed to be perfectly elastic.

This region can also be parallelized as the velocity and the coordinates are already calculated.

```
            // Checking for collisions and applying Newton physics for momentum
#pragma omp parallel for num_threads(threadCnt) collapse(2)
            for (int i = 0; i < body_cnt; i++)
            {
                for (int j = 0; j < body_cnt; j++)
                {
                    if (i == j)
                        continue;

                    double disX = abs(coord[i][0] - coord[j][0]);
                    disX *= disX;
                    double disY = abs(coord[i][1] - coord[j][1]);
                    disY *= disY;
                    double disZ = abs(coord[j][2] - coord[i][2]);
                    disZ *= disZ;

                    double dis = disX + disY + disZ;
                    dis = sqrt(dis);
                    if (dis < minDis && i < j)
                    {

                        swap(velocity[i][0], velocity[j][0]);

                        swap(velocity[i][1], velocity[j][1]);

                        swap(velocity[i][2], velocity[j][2]);
                    }
                }
            }
```

Finalizing The Results

After all the calculations, we store the coordinates of all the particles in a Coordinates.txt file after every 100 time step.

Finally we measure the amount of time taken to process everything and display the results, for comparison between different system specifications.

```
        // Displaying progress after every 100 iterations
        if (n % 100 == 0)
        {
            cout << n << "\n";
            res << "Iteration : " << n / 100 << "\n";
            for (int j = 0; j < body_cnt; j++)
            {
                res << coord[j][0] << " " << coord[j][1] << " " << coord[j][2] << "\n";
            }
        }
    }

    // Closing the output file
    res.close();

    // Displaying the total elapsed time
    end = omp_get_wtime();
    printf("Work took %f seconds\n", end - start);
```

Results

```
SYSTEM CONFIGURATON :
OS: windows 11
no of cores: 6
no of processors: 12
clock speed: 2.60 GHz

num of threads=1
Time required for each step: 4.092125 seconds
Total simulation time: 31520.245 seconds

num of threads=2
Time required for each step: 2.233125 seconds
Total simulation time: 17156.367 seconds

num of threads=4
Time required for each step: 1.117375 seconds
Total simulation time: 10064.942 seconds

num of threads=6
Time required for each step: 0.85350 seconds
Total simulation time: 8940.997 seconds
```

Graphic Visualiser

Using MATLAB

Reading the data from txt file

```
close all;
clear all;
clc;
filepath = 'Coordinates.txt';   %Coordinates of body path
fileId = fopen(filepath, 'r');  %Opening the file
line = fgetl(fileId);
w = str2double(regexp(line, '[\d.]+', 'match'));          %Width of the file
line = fgetl(fileId);
l = str2double(regexp(line, '[\d.]+', 'match'));          %Length of the file
line = fgetl(fileId);
d = str2double(regexp(line, '[\d.]+', 'match'));          %Depth of the file
line = fgetl(fileId);
N = str2double(regexp(line, '[\d.]+', 'match'));          %Number of bodies
line = fgetl(fileId);
r = str2double(regexp(line, '[\d.]+', 'match'));          %Radius of a body
line = fgetl(fileId);
t_step = str2double(regexp(line, '[\d.]+', 'match'));     %Time Step
line = fgetl(fileId);
num_step = str2double(regexp(line, '[\d.]+', 'match'));    %Number of time steps or iterations
data = zeros(3, N, num_step);
C = rand(N, 3); %Random Color for each ball
```

Opening the text file that has the coordinates of the particles

Reading individual lines from the text file and searching for numerical values in the line using regex expressions.

Using random number generator to randomize the colour of the particles in the plot.

Plotting the Many Bodies

```
figure("Name", "Many Body Graphics", 'units','normalized','outerposition',[0 0 1 1]);
for i = 1:num_step
    clf;   %clears the previous iteration plotted bodies of the figure
    line = fgetl(fileId);
    iter_num = str2double(regexp(line, '[\d.]+', 'match'));    %Reads the Iteration number

    coord = fscanf(fileId, '%f %f %f', [3, N*num_step]);%Reads all the Coordinates of the N bodies

    %plotting all the n bodies of the ith iteration
    scatter3(coord(1, :), coord(2, :), coord(3, :), r*70, C, 'filled');
    hold on;
    grid on;
    xlabel('x');
    ylabel('y');
    zlabel('z');
    title(["Many Bodies Positions at iteration = ",num2str(i)], "Color", 'r');
    axis([0, w, 0, l, 0, d]);
    hold off;
    [illegible]
image frame
    drawnow();

    data(:, :, i) = coord; %Storing the data for every time step in one 3d matrix
    disp("Iteration "+int2str(i)+" Data Loaded");
end
```

Plots all the N bodies of the current iteration

Video demonstration of Particles movement

```
myVideo = VideoWriter('ManyBodyGraphical');
myVideo.FrameRate = 10;
open(myVideo);
writeVideo(myVideo, currframe);
close(myVideo);
fclose(fileId);
```

Video writer which creates video

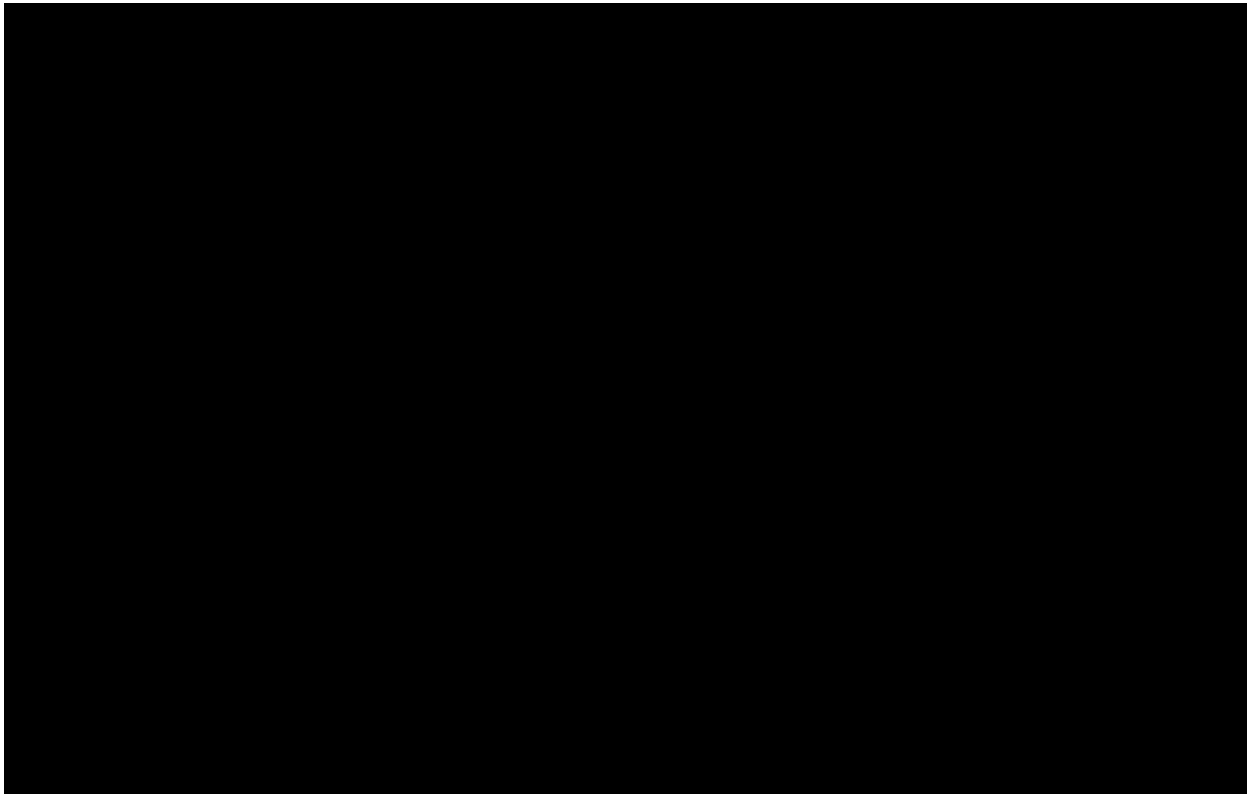
Controls frame rate of the video

Adds current frame of each iteration to video file

- Video will be stored in the same folder as the code in the format of .avi

Graphic Visualiser Output

- The video below shows the sample graphic visualizer output of many body oscillations simulated in MATLAB



Conclusion

- Many body collisions were implemented in a parallel environment which optimized the code and reduced the execution time as expected from an openmp environment.
- The scalability of threads proved in reducing the time complexity with parallel utilization of resources .
- Laws of physics and observations on collisions and their study could easily be modelled for large number of bodies with less execution time through openmp environment . Openmp has lot of processing capabilities and its parallel features are very useful in optimizing real life applications.

Link to Code and Other Files

Drive Link:
https://drive.google.com/drive/folders/1cLLeRcXoEvi9opQKnB3dcUGrlET4MLhJ

GitHub Link:
https://github.com/rlokesh2002/ManyBodySimulation.git


Thank you!