`

High Performance Computing

Project

Submitted by

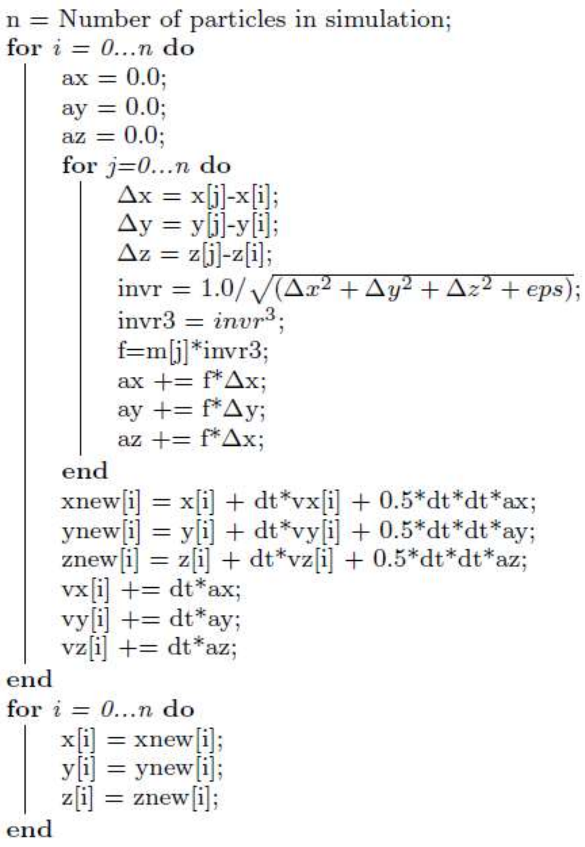
Shyam Sundar Vasu – 62719

Problem Description

The goal of this project is to write a C programm to simulate the motion of particles under gravitational force and parallelize it using MPI, on which weak and strong parallel scalability tests are to be carried out on the TUBAF cluster.

Methodology

Each particle is assigned a random initial position and velocity, which are advanced forward in time according to the provided algorithm shown below



```
n = Number of particles in simulation;
for $i$ = $0...n$ do
    ax = 0.0;
    ay = 0.0;
    az = 0.0;
    for $j$=$0...n$ do
        Δx = x[j]-x[i];
        Δy = y[j]-y[i];
        Δz = z[j]-z[i];
        invr = 1.0/$\sqrt{(\Delta x^2 + \Delta y^2 + \Delta z^2 + eps)}$;
        invr3 = $invr^3$;
        f=m[j]*invr3;
        ax += f*Δx;
        ay += f*Δy;
        az += f*Δx;
    end
    xnew[i] = x[i] + dt*vx[i] + 0.5*dt*dt*ax;
    ynew[i] = y[i] + dt*vy[i] + 0.5*dt*dt*ay;
    znew[i] = z[i] + dt*vz[i] + 0.5*dt*dt*az;
    vx[i] += dt*ax;
    vy[i] += dt*ay;
    vz[i] += dt*az;
end
for $i$ = $0...n$ do
    x[i] = xnew[i];
    y[i] = ynew[i];
    z[i] = znew[i];
end
```

The updated positions are to be written to a text file for visualization.

Implementation

In the parallel program, type defined struct are used for position, velocity and acceleration to facilitate simplification and understanding the flow as they all have x, y, z components. Also, they are defined as MPI datatypes for ease of parallelization.

Number of particles (Np), total time of simulation (total_time) and time step (dt) are taken as input through command line arguments for flexibility of the code for scalability testing.

The whole programm is divided into parts as functions to enhance clarity of workflow and to ease debugging. One separate function – "P_r" in the code is used to calculate send counts and displacements based on given number of particles and number of processes, in-order to dived the number of particles equally/unequally among the processors if they are not divisible by total number of processors used.

Position and velocity arrays are initialized with random values within the defined range by a function – "generate_pos_vel" at root processor before the start of simulation.

In the main simulation function- "sim", position and velocity arrays are broadcasted from root processor in-order to avoid start of simulation before its initialization. Followed by this time loop for n number of iterations is started. Inside this for loop three functions – "cal_acceleration", "cal_vel" and "cal_position" are called in sequence by each processor to calculate acceleration, velocity and position of particles assigned to that particular processor(rank). After this, all the updated parts of position and velocity arrays are gathered to the same array for clarity using "MPI_Allgatherv", in-order to gather uneven number of entities in case of odd number of particles.

At root process, positions at each time step is written to a text file which is later used for visualization. "MPI_Wtime" is used to calculate the computation time of the parallelized code.

Workflow

The boxed region shows the parallelized part of the programm.

Command line arguments – Np, total_time, dt

↓

Structs and data type definition for position, velocity and acceleration
in C and MPI

↓

Get MPI rank and size

↓

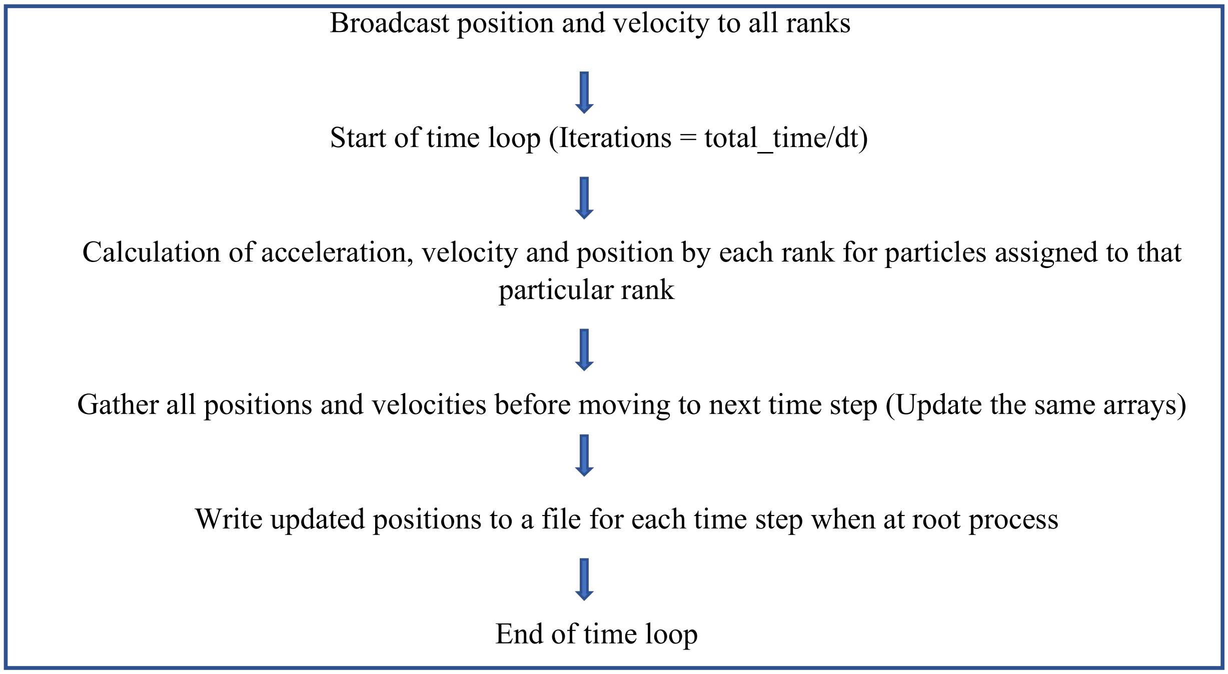
Get send counts and displacements for given number of particles and processors

↓

Generate initial position and velocities for particles

↓

`



Broadcast position and velocity to all ranks

Start of time loop (Iterations = total_time/dt)

Calculation of acceleration, velocity and position by each rank for particles assigned to that particular rank

Gather all positions and velocities before moving to next time step (Update the same arrays)

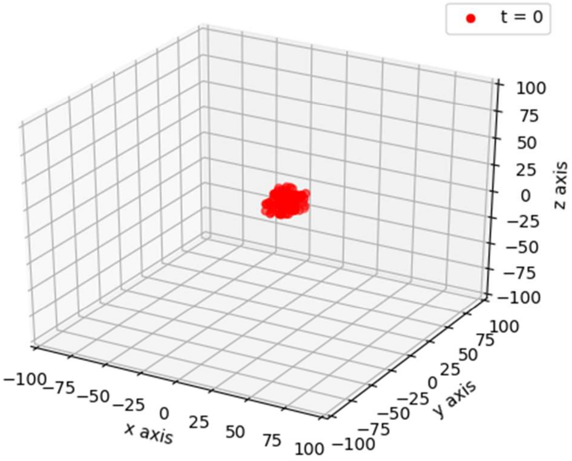
Write updated positions to a file for each time step when at root process

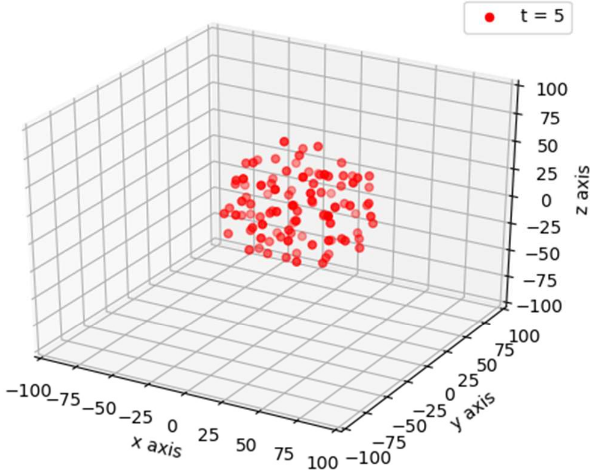
End of time loop

Visualization:

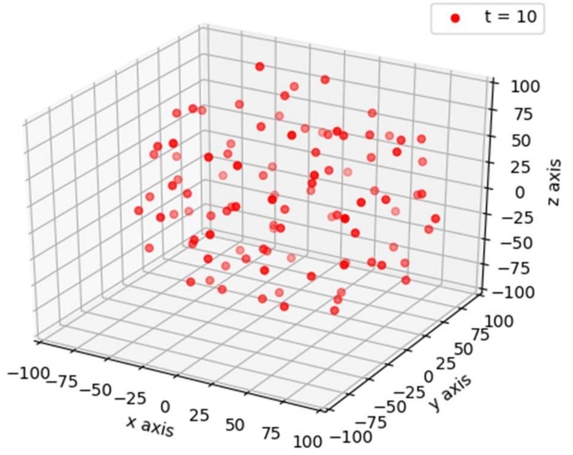
The output text file with positions of particles at each time step is imported in a python code to visualize the simulation results with the help of matplotlib 3D scatterplot animation. It is generated by modifying a 3D scatter plot with updated position co-ordinates.

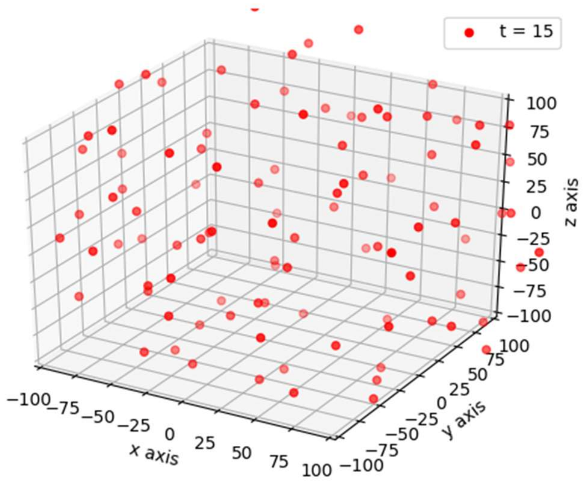
The following figures shows the position of 100 particles at different time frames.





`





Results and Discussion

Strong Parallel Scalability:

For Strong parallel scalability test the problem size is fixed to

Number of particles = 240
Total simulation time = 60 s

Simulations are carried out for different time steps with increasing number of processors from 1-12 and computational time is documented as shown below

<table>
<tr><th colspan="10">Strong Scalabiltiy | Numer of particles = 240 | t = 60s</th></tr>
<tr><th>No. of proc</th><th>dt=1</th><th>dt = 0.5</th><th>dt=0.25</th><th>dt=0.1</th><th>dt = 0.075</th><th>dt=0.05</th><th>dt=0.025</th><th>dt=0.02</th><th>dt=0.01</th></tr>
<tr><td>1</td><td>0.487989</td><td>0.962063</td><td>1.9132</td><td>4.779732</td><td>6.378348</td><td>9.617136</td><td>19.17927</td><td>23.977663</td><td>47.98928</td></tr>
<tr><td>2</td><td>0.276915</td><td>0.548467</td><td>1.082278</td><td>2.668634</td><td>3.464655</td><td>5.113607</td><td>10.042817</td><td>12.440547</td><td>24.57652</td></tr>
<tr><td>4</td><td>0.166978</td><td>0.320639</td><td>0.63694</td><td>1.595537</td><td>2.158738</td><td>3.057785</td><td>6.088053</td><td>7.551712</td><td>14.49783</td></tr>
<tr><td>6</td><td>0.127593</td><td>0.249715</td><td>0.49662</td><td>1.24305</td><td>1.709866</td><td>2.513844</td><td>4.75211</td><td>5.939037</td><td>11.89993</td></tr>
<tr><td>8</td><td>0.109997</td><td>0.217711</td><td>0.437057</td><td>1.117326</td><td>1.491642</td><td>2.220326</td><td>3.9455</td><td>5.133167</td><td>10.58133</td></tr>
<tr><td>10</td><td>0.099553</td><td>0.196945</td><td>0.392488</td><td>0.906809</td><td>1.294149</td><td>1.895078</td><td>3.750391</td><td>4.900631</td><td>9.183856</td></tr>
<tr><td>12</td><td>0.094669</td><td>0.182432</td><td>0.366763</td><td>0.900453</td><td>1.207511</td><td>1.795263</td><td>3.495446</td><td>4.417833</td><td>8.642253</td></tr>
</table>

This particular number of particles is chosen and only these number of processors are used, as it was fairly enough number of particles and it is only equally divisible by 1,2,4,6,8,10 and 12 number of processors.

`

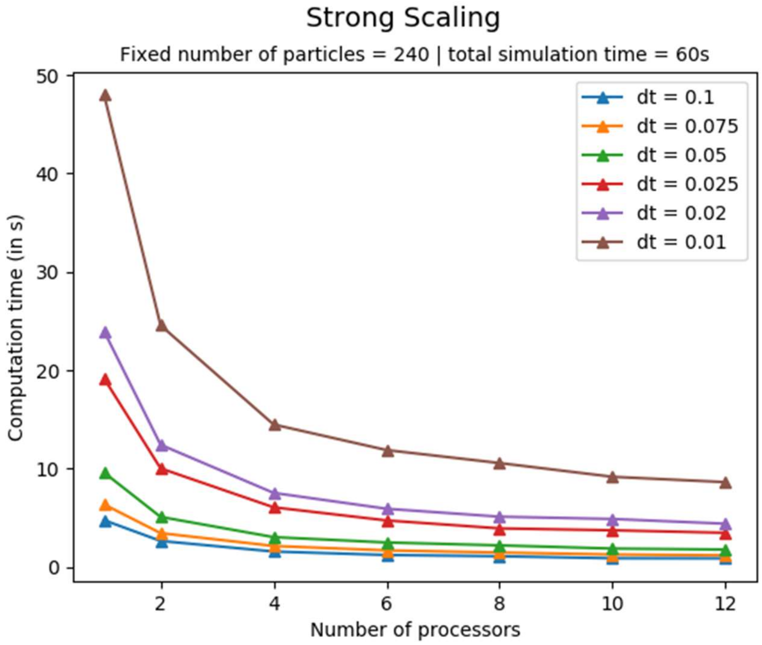


Fig 1. Strong Scaling

It can be observed that computational time is efficiently reduced for smaller time step values (i.e. lesser dt value) and also there is a jump in computational time from dt 0.02 to 0.01. So, from the plot above it is better to chose dt 0.02 to retain relatively high accuracy with reasonable computational time. Strong scaling efficiency can be calculated from the ratio of time taken for computation by 1 processor to product of number of processors and its computational time.

Strong scaling efficiency = $(T_1 / (Np * T_{Np})) * 100$ %

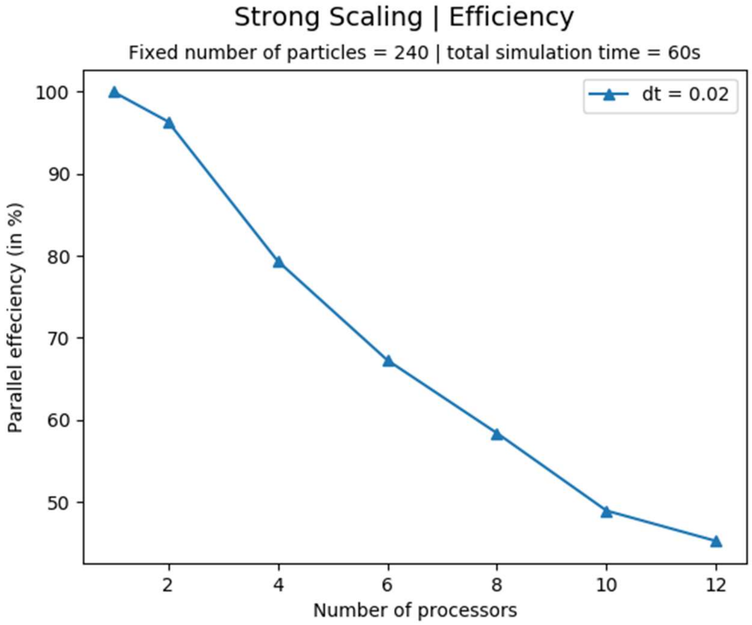


Fig 2. Strong scaling efficiency for dt 0.02

`

Weak Parallel Scalability:

For weak parallel scalability the problem size is increased in such way that number of particles assigned for each processor remains constant with increase in number of processors.

Number of particles per processor = 100
Total simulation time = 60 s

Weak Scalability | Number of particles/proc = 100 | t = 60s										
No. of proc	particles	dt=1	dt = 0.5	dt=0.25	dt=0.1	dt = 0.075	dt=0.05	dt=0.025	dt = 0.02	dt=0.01
1	100	0.130562	0.253452	0.493537	1.225528	1.494257	2.424117	4.461837	5.964859	12.03569
2	200	0.210091	0.411293	0.799119	1.975968	2.420639	3.797426	7.229393	9.197696	18.09143
3	300	0.29428	0.574633	1.155501	2.823826	3.459552	5.338583	10.24734	13.01607	26.00485
4	400	0.385265	0.757219	1.508126	3.536067	4.4631	6.88733	13.38353	17.10818	33.61238
5	500	0.462632	0.906199	1.83839	4.343585	5.568281	8.65993	16.89947	21.37725	42.64169
6	600	0.558263	1.109542	2.189637	5.232593	6.747563	10.27965	20.11635	25.35428	50.6852
7	700	0.638482	1.265276	2.520714	5.932547	7.817315	11.86426	23.43652	29.24557	58.55172
8	800	0.723294	1.44187	2.778811	6.682416	8.7708	13.51314	26.68541	33.44875	66.3014
9	900	0.815452	1.619831	3.140948	7.541545	9.907737	15.06359	29.67416	37.11636	74.57325
10	1000	0.89957	1.786882	3.440467	8.350048	10.92605	16.49208	32.57385	41.01411	82.27843
11	1100	0.990155	1.950994	3.734051	9.056939	11.92143	17.96435	35.70943	44.50628	89.04724
12	1200	1.060357	2.117478	4.10245	9.813691	13.10475	19.38096	38.71195	48.51718	96.91127

Simulations are carried out for different time steps with scaled up number of particles in par with increasing number of processors and computational time is documented as shown in the table above.

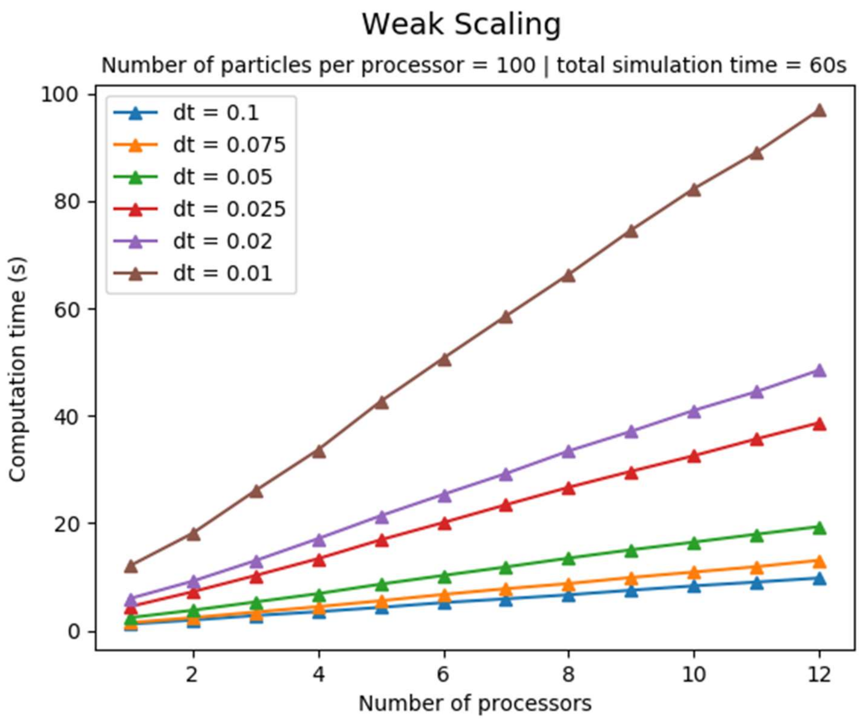


Fig 3. Weak Scaling

`

It is observed that the computational time is increasing linearly with increase in number of particles as expected and similar jump in computational time from time step 0.02 to 0.01 is noted. Even though this programm is capable of simulating unevenly distributed number of particles to each processor, for efficient parallelization number of particles should be equally divisible among the chosen number of processors.