HPP Assignment 3: Simulating the gravitational N-body problem

Oscar Jacobson

December 29, 2022

Abstract

In this report a simulation of free moving particles is carried out using the N-body problem formulation and the symplectic Euler method. The problem as well as the implementation and serial optimization in C code is described. The assignment was carried out at Uppsala university as a part of the course High Performance Programming.

1 The Problem

In this assignment the N-body problem of multiple particles was to be simulated. The N-body problem is a set of equations governing the movement and interactions between free moving bodies in an empty space. These equations are infeasible to solve by hand as the complexity grows exponentially with the amount of interacting bodies. Using computer simulations though this is a much more feasible task.

The equations used in this assignment are:

$$F_i = -Gm_i \sum_{j=0, i\neq j}^{N-1} \frac{m_j}{(r_{ij} + \epsilon_0)} \boldsymbol{r}_{ij} \tag{1}$$

Where G is a gravitational constant, m_j, m_i are individual particle masses r_{ij} is the distance between particles, ϵ_0[1] is a small constant and $\boldsymbol{r}_{ij}$ is:[2]

$$\boldsymbol{r}_{ij} = (x_i - x_j)\boldsymbol{e}_x + (y_i - y_j)\boldsymbol{e}_y \tag{2}$$

[1] ϵ_0 is used to counteract instabilities in the simulation and put a theoretical cap on calculated forces

[2] G is set to $\frac{100}{N}$, ϵ_0 is set to 10^{-3} and Δt is set to10^{-5}

The computer calculations are then simulated using the $SymplecticEuler$ formula for updating:

$$a_i^n = \frac{F_i^n}{m_i} \tag{3}$$

$$u_i^{n+1} = u_i^n + \Delta t a_i^n \tag{4}$$

$$x_i^{n+1} = x_i^n + \Delta t u_i^{n+1} \tag{5}$$

Where Δt is the step size a is acceleration u is velocity and x is postion. The complexity of computing the forces on all N particles grows exponentionally with complexity $O(n^2)$.

The code implemented to solve this problem is written in C, called galsim.c and is compiled to be named galsim. Using given input data, in binary form, as well as reference output data for specific step- numbers and sizes the code could be implemented.

The form of the input data is six floating point numbers of type double for every particle. The six numbers represent:

1. Position in x plane
2. Position in y plane
3. Particle mass
4. Velocity in x plane
5. Velocity in y plane
6. Brightness

Brightness is not used in any of the calculations but has to be preserved for all the particles throughout the simulations. The simulation takes place in a 1 X 1 dimension (The x and y plane) with a maximum value of 1 and a minimum value of 0.

2 The solution

The code used to simulate the N-body problem is appended in appendix A. The code takes five input variables:

1. An integer number N, for the amount of particles simulated

2. A string of the name of the infile used to read input values. (Has to be in directory "./input_data/")

3. An integer number stp, for the amount of time steps to simulate.

4. A double float number for the size of the time step.

5. A 1 or 0 signifying to turn graphics on or of.

Example: *./galsim 3000 ./input_data/ellipse_N_03000.gal 100 0.00001 0*

The code have two main parts. One part for function and variable definitions and one *main()* function for executing the code.
The first part initializes all variables used in the code. Defines a structure called *particle* which stores the six double float numbers mentioned in section 1 as well as a structure *fxy* which stores two double float numbers that will be used later to store forces in x and y directions. [1] The code then defines two functions with function handles *force_calc* and *pos_update*.
force_calc takes two particle structures as input variables, calculates the forces on the first particle caused by the second, and outputs the forces in x and y directions in a fxy structure defined above.

pos_update takes a pointer to a particle structure and two double float numbers as input values. The two numbers represent the force on the particle and using those numbers the function calculates the acceleration, new velocity and new position of the particle and updates these values using the pointer to particle structure.

The *main()* function in the program takes five (Six if counting the ./galsim) input variables and only runs if this is true. If not an error message is written out in the command prompt and the user is to try again. Using the double pointer char**argv as input values to main the input variables can be scanned from stdin and used in the program. Using argv[1 trough 5] the input values N, infile (Infile string), stp (Amount of timesteps), delta (Timestep size) and graph (Toggle graphics) are all given values.

The program then opens and reads the input file to particles structures, using FILE fopen and fread in a for loop that is N times long. The praticles structures get initialized as Particles[0 through N-1] in a function called *read1*.

Two arrays of N pointers to double numbers are then initialized using malloc called *accx and *accy. Two double float numbers sumx and sumy are also initialized as well as a *fxy* struct called *ftot*. [2] Three nested for loops are now used to

calculate the forces on each particle. The first loop calculates and sums the forces of all $j \neq i$ particles on particle i using *force_calc*. The second loop repeats the first loop for every particle i and stores the summations in the i:th value of the arrays **accx* and **accy*. The third loop uses the calculated forces to update the positions of all particles using *pos_update* and repeats the cycle for *stp* times (The amount of time steps).

The final values of the particles structure are then written to a binary file called results.gal using FILE, fopen, and fwrite defined in a function called *write1*.

Lastly the arrays *accx* and *accy* are freed together with the *particles* structure.

A visual representation of the first algorithm *galsim*:

Algorithm 1 *Non optimized Galsim*

```
for n steps do
    for i is [1-N] do
        for j is [1-N] do
            if i != j then
                force_calc(particle[i], particle[j])
            end if
        end for
    end for

    for i is [1-N] do
        pos_update(particle[i], Fx[i], Fy[i])
    end for
end for
```

2.1 Possible improvements

The force of particle i on j is equal to the negative force of j on i. Using this at least half of the calculations can be inferred from previous calculations. This would require some form of N * N long array to store the forces for all of the individual calculations.
pos_update() could use a pointer to iterate over and update all particle positions with one function call.
force_calc can be implemented to take one particle and return the sum of force from the other particles, might be beneficial depending on implementation beacuse of the reduced amount of required function calls. Taking this even further

force_calc() could be implemented to only require one function call per time-step.

This would simplify the algorithm to:

Algorithm 2 *Serially optimized Galsim*

```
for n steps do
    force_calc(particles)
    pos_update(particles, Fx, Fy)
end for
```

Which would greatly decrease the number of function calls. I am also aiming to reduce the amount of calculations done by *force_calc()* as well as optimizing by reducing multiplication and division where not necessary.

2.2 Optimization

Above there are two main ideas for optimization mentioned. These were carried out in two steps and will be referenced in the two different code files, *galsim* (original), *galsimCallOpt* (Reduced function calls), *galsimopt* (Reduced function calls and reduced calculations). All of my code is appended in Appendix A
For *galsimCallOpt* the functions *force_calc* and *pos_update* was written to take all particles as input and give the result for that time step iteration in one call.
Looking something like:

Algorithm 3 *force_calc* in *galsimCallOpt*

```
for i in [1-N] do
    sum = 0; (Reset forces for every particle i)
    for j in [1-N] do
        if i != j then
            Calculate distance(i, j)
            Calculate force(i, j)
            sum = sum + force(i, j)
        end if
    end for
    Save total forces on i in a vector
end for
```

For *galsimopt* the *force_calc* function was optimized to reduce the force calculations in every iteration. This was done by initializing two N long arrays called *ftotx* and *ftoty*, utilizing the fact that the force of particle *i* on *j* is equal to the negative

force of particle *j* on *i* in that iteration. Important here is that *force_calc* now only sums the forces in the x and y plane and does no extra division calls to calculate the acceleration. This is done in the first step of *pos_update* to ensure only one division (by the particle mass to get the acceleration) is needed for every iteration. The algorithm for *galsimopt*:

Algorithm 4 *force_calc* in *galsimopt*

```
for i in [1-N] do
    for j in [i-N] do
        (Notice how j goes from i to N)
        if i != j then
            Calculate distance(i, j)
            Calculate force(i, j)
            Add force(i, j) at point [i] in ftotx
            Add -force(i, j) at point [j] in ftoty
        end if
    end for
end for
```

3 Performance and discussion

The *galsim* code runs with no leaks in valgrind and clears check_A3 on arrhenius.it.uu.se.

The time complexity of the code should work out to be Big O (N^2). The calculations in the code are done N times for the N particles and repeated n times for the amount of time steps. n is linear or non-changing and will not be significant on large scales. A simple complexity calculation then gives:
n x N (Position updates) + N x N (Force calculations) => n*N + N^2 =>(at large N)=> N^2 => Big O(N^2). Figure 1 shows this to be true.

Regarding execution time, table 1, 2 and 3 shows that the performed optimizations have a real impact on code execution time. *galsimopt* is clearly faster for all optimization flags. Worth noting though is that the difference in performance for the -Ofast flag is lesser than the performance gain for the other optimization flags. This could very well be because of the fact that a lot of the possible "manual" code optimizations are already taken into account when using the -Ofast flag, especially inefficiency in loop construction and/or making too many calls to a function etc. What I believe is causing the slight speedup in the optimized -Ofast column is the reduction in multiplication calls, which are slower than addition calls.

When not using the -Ofast flag, the speedup is significant. When optimizing I made the claim that the optimization would reduce the calculations needed by a factor of 2. The non -Ofast-flags are in some cases almost two times faster than the non optimized code which points to my optimization being somewhat successful.

4 Results

Results from running galsim:
time ./galsim 3000 ./input_data/ellipse_N_03000.gal 100 0.00001 0

Flag	Real	User	Sys
None	1m52.266s	1m52.261s	0m0.003s
O1	1m48.069s	1m48.066s	0m0.003s
O2	1m45.648s	1m45.617s	0m0.004s
O3	1m39.819s	1m39.810s	0m0.003s
Ofast	0m4.080s	0m4.079s	0m0.001s

Table 1: Time for running galsim with different optimization flags.

Results from running galsimCallOpt:
time ./galsimCallOpt 3000 ./input_data/ellipse_N_03000.gal 100 0.00001 0

Flag	Real	User	Sys
None	1m50.841s	1m50.826s	0m0.006s
O1	1m39.922s	1m39.912s	0m0.009s
O2	1m45.264s	1m45.249s	0m0.0012s
O3	1m41.653s	1m41.632s	0m0.006s
Ofast	0m3.960s	0m3.956s	0m0.003s

Table 2: Time for running galsimCallOpt with different optimization flags.

Results from running galsimopt:
time ./galsimopt 3000 ./input_data/ellipse_N_03000.gal 100 0.00001 0

Flag	Real	User	Sys
None	1m9.526s	1m9.520s	0m0.003s
O1	0m54.781s	0m54.773s	0m0.004.s
O2	0m49.888s	0m49.885s	0m0.001s
O3	0m49.568s	0m49.566s	0m001.s
Ofast	0m3.884s	0m3.881s	0m0.003s

Table 3: Time for running galsimopt with different optimization flags.

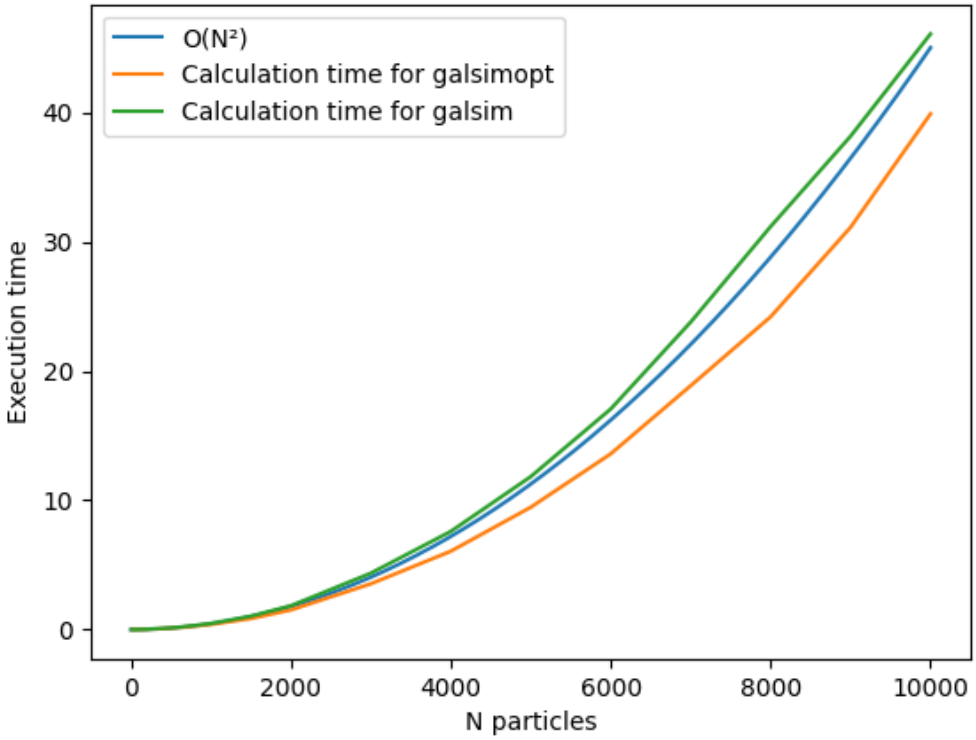

Figure 1: Plot of execution time vs N particles with -Ofast flag

I appendix listas en del program och större matriser som inte tillhör löpande text.

A Code

galsim.c

```
#include "graphics.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
 
 
 
 

/* Simulate galaxies */
/*Sources: https://stackoverflow.com/questions/46615620/c-
    structures-initializing-using-for-loop,    */

int N;
char * infile;
int stp;
float delta;
int graph;
int quit;
double G;
double epsil = 1.0/1000.0;
const float circleRadius=0.0025, circleColor=0;
const int windowWidth=1500;
float L=1, W=1;
 
 

typedef struct part{     /*Define class*/
    double Posx, Posy, Mass, Velx, Vely, Bright;
}particle;

typedef struct fxy{
    double fx, fy;
}f;
 
 
 

f force_calc(struct part particlesi, struct part particlesj){
           /*Takes two particles as input and returns the force
```

```
    in x and y direction*/
41      struct fxy f;
42      double Fx;
43      double Fy;
44      double distx = particlesi.Posx -particlesj.Posx;
45      double disty = particlesi.Posy -particlesj.Posy;
46      double rad = sqrt(distx*distx + disty*disty);
                /*Radius*/
47
48      /*printf("Rad %f\n", rad);
49      printf("Disty %f\n", disty);*/
50      /*printf("Pow %f\n", pow((rad+epsil),3));
51      printf("Pow %f\n", (rad+epsil)*(rad+epsil)*(rad+epsil));*/
52      Fx = -G*particlesi.Mass * particlesj.Mass/pow((rad+epsil),3)
        * distx;
53      Fy = -G*particlesi.Mass * particlesj.Mass/pow((rad+epsil),3)
        * disty;
54
55      f.fx = Fx;
56      f.fy = Fy;
57      /*printf("Force calc\nFx = %f Fy = %f\n", f.fx, f.fy);*/
58      return(f);
59  }
60
61
62  void pos_update(struct part *particles, double ax, double ay){
                /*Takes pointer to particle and updates its
    position given acceleration in x and y plane*/
63      particles->Velx = particles->Velx + delta*ax;
64      particles->Vely = particles->Vely + delta*ay;
65      /*printf("Vx = %f Vy = %f\n", particles->Velx, particles->
    Vely);*/
66
67      double px = particles->Posx + delta*particles->Velx;
68      double py = particles->Posy + delta*particles->Vely;
           /*If particles fly out they get put on the edge of the 1
    x 1 box*/
69      if (px >= 1){
70          particles->Posx = 1;
71      }
72      else if(px <= 0){
73          particles->Posx = 0;
74      }
75      else{
76          particles->Posx = px;
77      }
78
79      if (py >= 1){
80          particles->Posy = 1;
```

```
81      }
82      else if(py <= 0){
83          particles->Posy = 0;
84      }
85      else{
86          particles->Posy = py;
87      }
88      /*printf("Posupdate\nPosx = %f Posy = %f\n", particles->Posx
        , particles->Posy);*/
89  }
90
91
92  void read1(char *infile, particle *particles){
93      FILE *file;
94      file = fopen(infile, "rb");
95      for (int i = 0; i < N; i++) {
96          fread(&particles[i].Posx, sizeof(double), 1, file);
97          fread(&particles[i].Posy, sizeof(double), 1, file);
98          fread(&particles[i].Mass, sizeof(double), 1, file);
99          fread(&particles[i].Velx, sizeof(double), 1, file);
100         fread(&particles[i].Vely, sizeof(double), 1, file);
101         fread(&particles[i].Bright, sizeof(double), 1, file);
102     }
103     fclose(file);
104 }
105
106 void write1(particle *particles){
107     FILE *outfile;
108     outfile = fopen("result.gal", "wb");
109     for (int i = 0; i < N; i++) {
110         fwrite(&particles[i].Posx, sizeof(double), 1, outfile);
111         fwrite(&particles[i].Posy, sizeof(double), 1, outfile);
112         fwrite(&particles[i].Mass, sizeof(double), 1, outfile);
113         fwrite(&particles[i].Velx, sizeof(double), 1, outfile);
114         fwrite(&particles[i].Vely, sizeof(double), 1, outfile);
115         fwrite(&particles[i].Bright, sizeof(double), 1, outfile)
        ;
116     }
117     fclose(outfile);
118 }
119
120
121
122 void graphics(particle *particles, float L, float W, float
        circleRadius, float circleColor){
123     ClearScreen();
124     for (int i = 0; i < N; i++){
125         DrawCircle(particles[i].Posx, particles[i].Posy, L, W,
        circleRadius, circleColor);
```

```
126     }
127     Refresh();
128     usleep(3000);
129 }
130
131
132
133
134 int main(int argc, char** argv){
135     if (6 != argc){
136         printf("Needs five input variables!\n");
137     }
138     else{
139         N = atoi(argv[1]);
140         char * infile = argv[2];
141         stp = atoi(argv[3]);
142         delta = atof(argv[4]);
143         graph = atoi(argv[5]);
144         G = 100.0/N;
145
146         printf("\n\nN = %d\n", N);
147         printf("Input file = %s\n",infile);
148         printf("Number of timesteps = %d\n",stp);
149         printf("Timestep = %f\n",delta);
150         printf("Graphics = %d\n\n",graph);
151
152
153         particle *particles = NULL;
154         particles = calloc(N,sizeof(particle));
155
156         read1(infile, particles);
157
158
159         /*for (int i = 0; i < N; i++) {
160             printf("Start\nParticle %d\nPosx %f\nPosy %f\nMass %
    f\nVelx %f\nVely %f\nBrightness %f\n\n",i+1,particles[i].Posx
    ,particles[i].Posy,particles[i].Mass,particles[i].Velx,
    particles[i].Vely,particles[i].Bright);
161         }*/
162
163
164
165         /*initialize acc som pointer pointer i,j?       https://
    www.tutorialspoint.com/how-to-dynamically-allocate-a-2d-array
    -in-c*/
166         double *accx = (double *)malloc(N * sizeof(double));
      /*Malloc memory for N size array ix */
167         double *accy = (double *)malloc(N * sizeof(double));
      /*Malloc memory for N size array iy */
```

```
168         double sumx, sumy;
169         struct fxy ftot;
170 
171 
172 
173         /*struct fxy a;
174         struct fxy b;
175         a = force_calc(particles[2], particles[0]);*/
                    /*Test specific particles*//*
176         b = force_calc(particles[2], particles[1]);
177         printf("Force on particle %d x: %f y: %f\n", 1, a.fx, a.
    fy);
178         printf("Force on particle %d x: %f y: %f\n\n", 1, b.fx,
    b.fy);*/
179 
180         if (graph == 1){
181             InitializeGraphics(argv[0],windowWidth,windowWidth);
182             SetCAxes(0,1);
183             printf("Hit q to quit.\n");
184         }
185 
186 
187         for (int n = 0; n < stp; n++){
188             for (int i = 0; i < N; i++){
189                 sumx = 0;
190                 sumy = 0;
191                 for (int j = 0; j < N; j++){
192                     if (i != j){
193                         ftot = force_calc(particles[i],
    particles[j]);                  /*Force calc to get force in x
    and y direction for particle i*/  /*If for a matrix: (*arr is
     first value, *arr+i*N is ith column(or row idk), *arr+i*N+j
    is jth element of ith row*/
194                         sumx = sumx + ftot.fx;
195                         sumy = sumy + ftot.fy;
196                     }
197                 }
198                 /*printf("Sum of x forces %f Sum of y forces %f\
    n\n", sumx, sumy);*/
199                 *(accx + i) = sumx/particles[i].Mass;
                                    /*sum accelerations in every
    column*/
200                 *(accy + i) = sumy/particles[i].Mass;
201             }
202             /*for (int i = 0; i < N; i++){
203                 printf("Force on particle %d x: %f y: %f\n", i,
    accx[i]*particles[i].Mass, accy[i]*particles[i].Mass);
204             }*/
205 
```

```
206             /* step all particles*/
207             for (int i = 0; i < N; i++){
208                 /*printf("Before pos update particle %d\n Posx %
    f Posy %f\n Velx %f Vely %f\n Accx %f Accy %f\n\n", i,
    particles[i].Posx, particles[i].Posy, particles[i].Velx,
    particles[i].Vely, accix[i], acciy[i]);*/
209                 pos_update(&particles[i], accx[i], accy[i]);
210                 if (graph == 1 && quit != 1){
211                     graphics(particles, L, W, circleRadius,
    circleColor);
212                     quit = CheckForQuit();
213                     if (quit == 1){
214                         FlushDisplay();
215                         CloseDisplay();
216                     }
217                 }
218 
219                 /*printf("After pos update particle %d\n Posx %f
     Posy %f\n Velx %f Vely %f\n Accx %f Accy %f\n\n", i,
    particles[i].Posx, particles[i].Posy, particles[i].Velx,
    particles[i].Vely, accix[i], acciy[i]);*/
220             }
221         }
222 
223 
224 
225         write1(particles);
226 
227 
228 
229         /*for (int i = 0; i < N; i++) {
230             printf("Finish\nParticle %d\nPosx %f\nPosy %f\nMass
    %f\nVelx %f\nVely %f\nBrightness %f\n\n",i+1,particles[i].
    Posx,particles[i].Posy,particles[i].Mass,particles[i].Velx,
    particles[i].Vely,particles[i].Bright);
231         }*/
232 
233 
234         free(accx);
235         free(accy);
236         free(particles);
237     }
238     return(0);
239 }
```

galsimCallOpt.c

```
1 #include "graphics.h"
2 #include <math.h>
3 #include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>
```

```c
/* Simulate galaxies */
/*Sources: https://stackoverflow.com/questions/46615620/c-structures-initializing-using-for-loop,    */

/*Main*/
int N;
char * infile;
int stp;
float delta;
int graph;
int quit;
double G;

typedef struct part{     /*Define class*/
    double Posx, Posy, Mass, Velx, Vely, Bright;
}particle;


/*Graph*/
const double epsil = 1.0/1000.0;
const float circleRadius=0.0025, circleColor=0;
const int windowWidth=1500;
float L=1, W=1;

/*Force calc*/
typedef struct fxy{
    double fx, fy;
}F;

struct fxy f;
double Fx;
double Fy;
double distx;
double disty;
double rad;
double sumx, sumy;
struct fxy ftot;

/*Pos update*/
double x, y, Vx, Vy, px, py;
```

```
52
53
54
55
56
57
58
59 void force_calc(struct part *particles, double *ax, double *ay){
                /*Takes two particles as input and returns the
   force in x and y direction*/
60
61
62     /*printf("Rad %f\n", rad);
63     printf("Disty %f\n", disty);*/
64     /*printf("Pow %f\n", pow((rad+epsil),3));
65     printf("Pow %f\n", (rad+epsil)*(rad+epsil)*(rad+epsil));*/
66     for (int i = 0; i < N; i++){
67         sumx = 0;
68         sumy = 0;
69         for (int j = 0; j < N; j++){
70             if (i != j){
71                 distx = (particles + i)->Posx -(particles + j)->
   Posx;                                    /*Force calculations
   for i, j*/
72                 disty = (particles + i)->Posy -(particles + j)->
   Posy;
73                 rad = sqrt(distx*distx + disty*disty);
74                 Fx = -G*(particles + i)->Mass * (particles + j)
   ->Mass/pow((rad+epsil),3) * distx;
75                 Fy = -G*(particles + i)->Mass * (particles + j)
   ->Mass/pow((rad+epsil),3) * disty;
76                                         /*Make Fx and Fy into 2D
   -matrix sum later*/
77                 sumx = sumx + Fx;
                                            /*Add forces to the
   total force on i*/
78                 sumy = sumy + Fy;
79             }
80         }
81         *(ax + i) = sumx/(particles + i)->Mass;
                                           /*Total x-force from
   every particle on i gives its acceleration, stored in accx*/
82         *(ay + i) = sumy/(particles + i)->Mass;
                                           /*Total Y-force from
   every particle on i gives its acceleration, stored in accy*/
83     }
84     /*printf("Force calc\nFx = %f Fy = %f\n", f.fx, f.fy);*/
85 }
86
```

```
87
88  void pos_update(struct part *particles, double *ax, double *ay){
                /*Takes pointer to particle and updates its
    position given acceleration in x and y plane*/
89      for (int i = 0; i < N; i++){
90          x = *(ax + i);
91          y = *(ay + i);
92          Vx = (particles + i)->Velx;
93          Vy = (particles + i)->Vely;
94
95          particles[i].Velx = Vx + delta*x;
96          particles[i].Vely = Vy + delta*y;
97
98
99          px = (particles + i)->Posx + delta*Vx;
100         py = (particles + i)->Posy + delta*Vy;            /*If
    particles fly out they get put on the edge of the 1 x 1 box*/
101         if (px >= 1){
102             particles[i].Posx = 1;
103         }
104         else if(px <= 0){
105             particles[i].Posx = 0;
106         }
107         else{
108             particles[i].Posx = px;
109         }
110
111         if (py >= 1){
112             particles[i].Posy = 1;
113         }
114         else if(py <= 0){
115             particles[i].Posy = 0;
116         }
117         else{
118             particles[i].Posy = py;
119         }
120         /*printf("Posupdate\nPosx = %f Posy = %f\n", particles->
    Posx, particles->Posy);*/
121     }
122 }
123
124
125 void read1(char *infile, particle *particles){
126     FILE *file;
127     file = fopen(infile, "rb");
128     for (int i = 0; i < N; i++) {
129         fread(&particles[i].Posx, sizeof(double), 1, file);
130         fread(&particles[i].Posy, sizeof(double), 1, file);
131         fread(&particles[i].Mass, sizeof(double), 1, file);
```

```
132         fread(&particles[i].Velx, sizeof(double), 1, file);
133         fread(&particles[i].Vely, sizeof(double), 1, file);
134         fread(&particles[i].Bright, sizeof(double), 1, file);
135     }
136     fclose(file);
137 }
138 
139 void write1(particle *particles){
140     FILE *outfile;
141     outfile = fopen("result.gal", "wb");
142     for (int i = 0; i < N; i++) {
143         fwrite(&particles[i].Posx, sizeof(double), 1, outfile);
144         fwrite(&particles[i].Posy, sizeof(double), 1, outfile);
145         fwrite(&particles[i].Mass, sizeof(double), 1, outfile);
146         fwrite(&particles[i].Velx, sizeof(double), 1, outfile);
147         fwrite(&particles[i].Vely, sizeof(double), 1, outfile);
148         fwrite(&particles[i].Bright, sizeof(double), 1, outfile)
    ;
149     }
150     fclose(outfile);
151 }
152 
153 
154 
155 void graphics(particle *particles, float L, float W, float
    circleRadius, float circleColor){
156     ClearScreen();
157     for (int i = 0; i < N; i++){
158         DrawCircle(particles[i].Posx, particles[i].Posy, L, W,
    circleRadius, circleColor);
159     }
160     Refresh();
161     usleep(3000);
162 }
163 
164 
165 
166 
167 int main(int argc, char** argv){
168     if (6 != argc){
169         printf("Needs five input variables!\n");
170     }
171     else{
172         N = atoi(argv[1]);
173         char * infile = argv[2];
174         stp = atoi(argv[3]);
175         delta = atof(argv[4]);
176         graph = atoi(argv[5]);
177         G = 100.0/N;
```

```
178
179         printf("\n\nN = %d\n", N);
180         printf("Input file = %s\n",infile);
181         printf("Number of timesteps = %d\n",stp);
182         printf("Timestep = %f\n",delta);
183         printf("Graphics = %d\n\n",graph);
184
185
186         particle *particles = NULL;
187         particles = calloc(N,sizeof(particle));
188
189         read1(infile, particles);
190
191
192         /*for (int i = 0; i < N; i++) {
193             printf("Start\nParticle %d\nPosx %f\nPosy %f\nMass %
    f\nVelx %f\nVely %f\nBrightness %f\n\n",i+1,particles[i].Posx
    ,particles[i].Posy,particles[i].Mass,particles[i].Velx,
    particles[i].Vely,particles[i].Bright);
194         }*/
195
196
197
198         /*initialize acc som pointer pointer i,j?       https://
    www.tutorialspoint.com/how-to-dynamically-allocate-a-2d-array
    -in-c*/
199         double *accx = (double *)malloc(N * sizeof(double));
      /*Malloc memory for N size array ix */
200         double *accy = (double *)malloc(N * sizeof(double));
      /*Malloc memory for N size array iy */
201
202
203
204
205         /*struct fxy a;
206         struct fxy b;
207         a = force_calc(particles[2], particles[0]);*/
                  /*Test specific particles*//*
208         b = force_calc(particles[2], particles[1]);
209         printf("Force on particle %d x: %f y: %f\n", 1, a.fx, a.
    fy);
210         printf("Force on particle %d x: %f y: %f\n\n", 1, b.fx,
    b.fy);*/
211
212         if (graph == 1){
213             InitializeGraphics(argv[0],windowWidth,windowWidth);
214             SetCAxes(0,1);
215             printf("Hit q to quit.\n");
216         }
```

```
217
218
219
220         for (int n = 0; n < stp; n++){
221             force_calc(particles, accx, accy);                  /*
    Force calc to get force in x and y direction for particle i*/
      /*If for a matrix: (*arr is first value, *arr+i*N is ith
    column(or row idk), *arr+i*N+j is jth element of ith row*/
222
223
224
        /*printf("Sum of x forces %f Sum of y forces %f\n\n",
    sumx, sumy);*/
225
    /*for (int i = 0; i < N; i++){
226
        printf("Force on particle %d x: %f y: %f\n", i, accx[i]*
    particles[i].Mass, accy[i]*particles[i].Mass);
227
    }*/
228
    /*printf("Before pos update particle %d\n Posx %f Posy %f\n
    Velx %f Vely %f\n Accx %f Accy %f\n\n", i, particles[i].Posx,
     particles[i].Posy, particles[i].Velx, particles[i].Vely,
    accix[i], acciy[i]);*/
229             /* step all particles*/
230             pos_update(particles, accx, accy);
231
232             /*Update graphics*/
233             if (graph == 1 && quit != 1){
234                 graphics(particles, L, W, circleRadius,
    circleColor);
235                 quit = CheckForQuit();
236                 if (quit == 1){
237                     FlushDisplay();
238                     CloseDisplay();
239                 }
240             }
241             /*printf("After pos update particle %d\n Posx %f
    Posy %f\n Velx %f Vely %f\n Accx %f Accy %f\n\n", i,
    particles[i].Posx, particles[i].Posy, particles[i].Velx,
    particles[i].Vely, accix[i], acciy[i]);*/
242         }
243
244
245
246         write1(particles);
247
248
```

```
249
250         /*for (int i = 0; i < N; i++) {
251             printf("Finish\nParticle %d\nPosx %f\nPosy %f\nMass
    %f\nVelx %f\nVely %f\nBrightness %f\n\n",i+1,particles[i].
    Posx,particles[i].Posy,particles[i].Mass,particles[i].Velx,
    particles[i].Vely,particles[i].Bright);
252         }*/
253
254
255         free(accx);
256         free(accy);
257         free(particles);
258     }
259     return(0);
260 }
```

galsimopt.c

```
1  #include "graphics.h"
2  #include <math.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6
7
8
9
10
11 /* Simulate galaxies */
12 /*Sources: https://stackoverflow.com/questions/46615620/c-
       structures-initializing-using-for-loop,    */
13
14 /*Main*/
15 int N;
16 char * infile;
17 int stp;
18 float delta;
19 int graph;
20 int quit;
21 double G;
22
23 typedef struct part{     /*Define class*/
24     double Posx, Posy, Mass, Velx, Vely, Bright;
25 }particle;
26
27
28 /*Graph*/
29 const double epsil = 1.0/1000.0;
30 const float circleRadius=0.0025, circleColor=0;
31 const int windowWidth=1000;
```

```
32 float L=1, W=1;
33
34 /*Force calc*/
35 typedef struct fxy{
36     double fx, fy;
37 }F;
38
39 struct fxy f;
40 double Fx;
41 double Fy;
42 double distx;
43 double disty;
44 double rad;
45 double sumx, sumy;
46 struct fxy ftot;
47
48 /*Pos update*/
49 double ax, ay, Vx, Vy, px, py;
50
51
52
53
54
55
56
57
58
59 void force_calc(struct part *particles, double *fx, double *fy){
                /*Takes two particles as input and returns the
    force in x and y direction*/
60
61
62     /*printf("Rad %f\n", rad);
63     printf("Disty %f\n", disty);*/
64     /*printf("Pow %f\n", pow((rad+epsil),3));
65     printf("Pow %f\n", (rad+epsil)*(rad+epsil)*(rad+epsil));*/
66
67     for (int i = 0; i < N; i++){
68         for (int j = i+1; j < N; j++){
69             if (i != j){
70                 distx = (particles + i)->Posx -(particles + j)->
    Posx;                                   /*Force calculations
    for i, j*/
71                 disty = (particles + i)->Posy -(particles + j)->
    Posy;
72                 rad = sqrt(distx*distx + disty*disty);
73                 Fx = -G*(particles + i)->Mass * (particles + j)
    ->Mass/pow((rad+epsil),3) * distx;
74                 Fy = -G*(particles + i)->Mass * (particles + j)
```

```
    ->Mass/pow((rad+epsil),3) * disty;
                                                /*Make Fx and Fy into 2D
    -matrix sum later*/
                *(fx + i) += Fx;
                *(fy + i) += Fy;
                *(fx + j) += -Fx;
                *(fy + j) += -Fy;
            }
        }
    }
}


void pos_update(struct part *particles, double *fx, double *fy){
                /*Takes pointer to particle and updates its
    position given acceleration in x and y plane*/
    for (int i = 0; i < N; i++){
        ax = *(fx + i)/(particles + i)->Mass;
        ay = *(fy + i)/(particles + i)->Mass;
        Vx = (particles + i)->Velx;
        Vy = (particles + i)->Vely;

        particles[i].Velx = Vx + delta*ax;
        particles[i].Vely = Vy + delta*ay;


        px = (particles + i)->Posx + delta*Vx;
        py = (particles + i)->Posy + delta*Vy;            /*If
    particles fly out they get put on the edge of the 1 x 1 box*/
        if (px >= 1){
            particles[i].Posx = 1;
        }
        else if(px <= 0){
            particles[i].Posx = 0;
        }
        else{
            particles[i].Posx = px;
        }

        if (py >= 1){
            particles[i].Posy = 1;
        }
        else if(py <= 0){
            particles[i].Posy = 0;
        }
        else{
            particles[i].Posy = py;
        }
        *(fx + i) = 0;
```

```
119         *(fy + i) = 0;
120         /*printf("Posupdate\nPosx = %f Posy = %f\n", particles->
    Posx, particles->Posy);*/
121     }
122 }
123
124
125 void read1(char *infile, particle *particles){
126     FILE *file;
127     file = fopen(infile, "rb");
128     for (int i = 0; i < N; i++) {
129         fread(&particles[i].Posx, sizeof(double), 1, file);
130         fread(&particles[i].Posy, sizeof(double), 1, file);
131         fread(&particles[i].Mass, sizeof(double), 1, file);
132         fread(&particles[i].Velx, sizeof(double), 1, file);
133         fread(&particles[i].Vely, sizeof(double), 1, file);
134         fread(&particles[i].Bright, sizeof(double), 1, file);
135     }
136     fclose(file);
137 }
138
139 void write1(particle *particles){
140     FILE *outfile;
141     outfile = fopen("result.gal", "wb");
142     for (int i = 0; i < N; i++) {
143         fwrite(&particles[i].Posx, sizeof(double), 1, outfile);
144         fwrite(&particles[i].Posy, sizeof(double), 1, outfile);
145         fwrite(&particles[i].Mass, sizeof(double), 1, outfile);
146         fwrite(&particles[i].Velx, sizeof(double), 1, outfile);
147         fwrite(&particles[i].Vely, sizeof(double), 1, outfile);
148         fwrite(&particles[i].Bright, sizeof(double), 1, outfile)
    ;
149     }
150     fclose(outfile);
151 }
152
153
154
155 void graphics(particle *particles, float L, float W, float
    circleRadius, float circleColor){
156     ClearScreen();
157     for (int i = 0; i < N; i++){
158         DrawCircle(particles[i].Posx, particles[i].Posy, L, W,
    circleRadius, circleColor);
159     }
160     Refresh();
161     usleep(3000);
162 }
163
```

```
164
165
166
167 int main(int argc, char** argv){
168     if (6 != argc){
169         printf("Needs five input variables!\n");
170     }
171     else{
172         N = atoi(argv[1]);
173         char * infile = argv[2];
174         stp = atoi(argv[3]);
175         delta = atof(argv[4]);
176         graph = atoi(argv[5]);
177         G = 100.0/N;
178
179         printf("\n\nN = %d\n", N);
180         printf("Input file = %s\n",infile);
181         printf("Number of timesteps = %d\n",stp);
182         printf("Timestep = %f\n",delta);
183         printf("Graphics = %d\n\n",graph);
184
185
186         particle *particles = NULL;
187         particles = calloc(N,sizeof(particle));
188
189         read1(infile, particles);
190
191
192         /*for (int i = 0; i < N; i++) {
193             printf("Start\nParticle %d\nPosx %f\nPosy %f\nMass %
    f\nVelx %f\nVely %f\nBrightness %f\n\n",i+1,particles[i].Posx
    ,particles[i].Posy,particles[i].Mass,particles[i].Velx,
    particles[i].Vely,particles[i].Bright);
194         }*/
195
196
197
198         /*initialize acc som pointer pointer i,j?       https://
    www.tutorialspoint.com/how-to-dynamically-allocate-a-2d-array
    -in-c*/
199         double *ftotx = (double *)malloc(N * sizeof(double));
                    /*Malloc memory for N size array ax */
200         double *ftoty = (double *)malloc(N * sizeof(double));
                    /*Malloc memory for N size array ay */
201
202
203
204
205
```

```
206         /*struct fxy a;
207         struct fxy b;
208         a = force_calc(particles[2], particles[0]);*/
                    /*Test specific particles*//*
209         b = force_calc(particles[2], particles[1]);
210         printf("Force on particle %d x: %f y: %f\n", 1, a.fx, a.
    fy);
211         printf("Force on particle %d x: %f y: %f\n\n", 1, b.fx,
    b.fy);*/
212
213         if (graph == 1){
214             InitializeGraphics(argv[0],windowWidth,windowWidth);
215             SetCAxes(0,1);
216             printf("Hit q to quit.\n");
217         }
218
219
220
221         for (int n = 0; n < stp; n++){
222             force_calc(particles, ftotx, ftoty);
    /*Force calc to get force in x and y direction for particle i
    */  /*If for a matrix: (*arr is first value, *arr+i*N is ith
    column(or row idk), *arr+i*N+j is jth element of ith row*/
223
224
225
        /*printf("Sum of x forces %f Sum of y forces %f\n\n",
    sumx, sumy);*/
226
    /*for (int i = 0; i < N; i++){
227
        printf("Force on particle %d x: %f y: %f\n", i, accx[i]*
    particles[i].Mass, accy[i]*particles[i].Mass);
228
    }*/
229
    /*printf("Before pos update particle %d\n Posx %f Posy %f\n
    Velx %f Vely %f\n Accx %f Accy %f\n\n", i, particles[i].Posx,
     particles[i].Posy, particles[i].Velx, particles[i].Vely,
    accix[i], acciy[i]);*/
230             /* step all particles*/
231             pos_update(particles, ftotx, ftoty);
232
233             /*Update Graphics*/
234             if (graph == 1 && quit != 1){
235                 graphics(particles, L, W, circleRadius,
    circleColor);
236                 quit = CheckForQuit();
237                 if (quit == 1){
```

```
238                     FlushDisplay();
239                     CloseDisplay();
240                 }
241             }
242             /*printf("After pos update particle %d\n Posx %f
    Posy %f\n Velx %f Vely %f\n Accx %f Accy %f\n\n", i,
    particles[i].Posx, particles[i].Posy, particles[i].Velx,
    particles[i].Vely, accix[i], acciy[i]);*/
243         }
244
245
246
247         write1(particles);
248
249
250
251         /*for (int i = 0; i < N; i++) {
252             printf("Finish\nParticle %d\nPosx %f\nPosy %f\nMass
    %f\nVelx %f\nVely %f\nBrightness %f\n\n",i+1,particles[i].
    Posx,particles[i].Posy,particles[i].Mass,particles[i].Velx,
    particles[i].Vely,particles[i].Bright);
253         }*/
254
255         free(ftotx);
256         free(ftoty);
257         free(particles);
258     }
259     return(0);
260 }
```

References

[1] David C. Rankin. https://stackoverflow.com/questions/46615620/c-structures-initializing-using-for-loop. 2022-02-15.

[2] Chandu yadav. https://www.tutorialspoint.com/how-to-dynamically-allocate-a-2d-array-in-c. *Touturialspoint*, 2022-2-15.