ASSIGNMENT 3

REPORT : G 29

By AGNE Amadou, CHAMPDAVOINE Nicolas & BAILLEUX Madeleine

ASSIGNMENT 3

Content

Content _____ 2
The problem _____ 3
The solution _____ 4
Performance _____ 9

Nicolas made the major part of the code and the optimization and the graphic about performance. Amadou and Madeleine worked on the sympletic Euler's method and on the report.

ASSIGNMENT 3

The problem

For the Assignment 3 we had to implement a code that calculates the evolution of N particles in a gravitational simulation. The input arguments are:

- N, the number of particles
- nbsteps, the number of timesteps between time 0 (initial time) to time T (final time)
- Delta t is the value of one timestep
- Data by particle: position x,position y, mass, velocity x and velocity y. To test our code we have a set of input files with some input data of N particles.

So in our code we should have some parts:

- A part to open and read the binary file with input data so with this part of code we can use them in the rest of the code
- A part to calculate with formulas the force between i-particle and others particles, the acceleration, velocity and position at time n.
- A part to write the results in an output binary file with the new position x, position y, mass, velocity x,velocity y at time T(final time).
- Extra part to draw the graphic and see the galaxy evolution on the screen

The solution

- Initializing particles data with the input files

We have some input binary files .gal with data for N particles (position x, position y, mass, velocity x, velocity y). At the beginning of our code we open and read the binary file that we have chosen.

We define the structure “particle” with:

```
// Definition of particles by their positions,mass and velocities
typedef struct particule {
    double pos_x;
    double pos_y;
    double mass;
    double vel_x;
    double vel_y;
}*particule;
```

In argument the user has to enter the filename of the input binary file he wants to use. (with the number of particles N, the number of steps, the value of a time step ad if he wants to draw a graphic or not).

```
int main (int argc, char *argv[]){


   if (argc != 6){
       printf("Wrong number of arguments given. Write:%s nbr_of_star filename nsteps delta_t graphics_0/1", argv[0]);
       return -1;
   }

   const int N = atoi(argv[1]);
   const char* fileName = argv[2];
   const int nsteps = atoi(argv[3]);
   const double delta_t = atof(argv[4]);
   const int graphics = atoi(argv[5]);
//Definition of Epsilon0
   const double E0 = 0.001;
   particule particules[N];
```

We use the function read_doubles_from_file:

```
if (read_doubles_from_file(5*N, input, fileName) != 0){
    printf("Error reading file \n");
    return -1;

}
```

Now we associate each element of the binary file with a parameter of a particle:

```
// Initializing particules data with the input file
for(i = 0; i<N; i++){
    particules[i] = (struct particule *)malloc(sizeof(struct particule));
    particules[i]->pos_x = input[i*5 + 0];
    particules[i]->pos_y = input[i*5 + 1];
    particules[i]->mass  = input[i*5 + 2];
    particules[i]->vel_x = input[i*5 + 3];
    particules[i]->vel_y = input[i*5 + 4];
}
```

If we do a printf at this part we obtain:

```
file open
x0.500000
y:0.200000
mass:1.000000
vel_x:6.500000
vel_y:0.000000
x0.500000
y:0.800000
mass:1.000000
vel_x:-6.500000
vel_y:0.000000
```

(for the binary file circe_N_2.gal)

We have the position x,y the mass, the velocity x,y of each particles (here N=2).

- Evolution of parameters of each particles during number of steps

In this part we have the initial parameters for N particles of the galaxy. Now we have to calculate the new position and velocity after a number of steps with the value delta_t.

We are going to use these formulas and the sympletic Euler's method:

$$\begin{aligned} \mathbf{a}_i^n &= \frac{\mathbf{F}_i^n}{m_i}, \\ \mathbf{u}_i^{n+1} &= \mathbf{u}_i^n + \Delta t \mathbf{a}_i^n, \\ \mathbf{x}_i^{n+1} &= \mathbf{x}_i^n + \Delta t \mathbf{u}_i^{n+1}, \end{aligned}$$

We also need the F formula:

$$\mathbf{F}_i = -Gm_i \sum_{j=0, j\neq i}^{N-1} \frac{m_j}{(r_{ij}+\epsilon_0)^3}\mathbf{r}_{ij}.$$

So at the beginning of our code we declare the variables for the sympletic Euler's method.

```
int i, j, p;
double rij, cst_j, cord_x, cord_y;
double distancex, distancey;
double sum_Fx, sum_Fy, Ax, Ay;
```

And this is the code for the sympletic Euler's method:

```
const double Gdelta_t = (-100.0/n)*delta_t;
for (p=0; p<nsteps; p++) {
    double sum_Fx[N], sum_Fy[N];
    if (graphics == 1) ClearScreen();
    for (i=0; i<N; i++) {
        for (j=i; j<N; j++) {
            distancex = particules[i]->pos_x - particules[j]->pos_x;
            distancey = particules[i]->pos_y - particules[j]->pos_y;
            rij = sqrt(distancex*distancex + distancey*distancey);
            cst_j = 1.0 /((rij+E0)*(rij+E0)*(rij+E0));
            cord_x = cst_j * distancex;
            cord_y = cst_j * distancey;
            sum_Fx[i] += particules[j]->mass * cord_x;
            sum_Fy[i] += particules[j]->mass * cord_y;
            sum_Fx[j] -= particules[i]->mass * cord_x;
            sum_Fx[j] -= particules[i]->mass * cord_y;
        }
    }

    for (i=0; i<N; i++){
        particules[i]->vel_x += Gdelta_t * sum_Fx[i];
        particules[i]->vel_y += Gdelta_t * sum_Fy[i];
        particules[i]->pos_x += delta_t*particules[i]->vel_x;
        particules[i]->pos_y += delta_t*particules[i]->vel_y;
```

We use 2 loops with i and one loop with j. in the loop with j we are going to calculate the sum for the formula F. After in the first loop i we actualize values of particules[i] ->vel_x, particules[i]->vel_y, but we don't modify values of position x and y because we have to use them for others particles.

In the second loop i we integrate the values of velocity to have values of positions.

To actualize values we use sympletic Euler's formulas with the acceleration, the velocity and the position.

- Write an output binary file

We have to write an output binary file with new values of each parameters for each particles.

```
//We put in output the result of Euler's method and put output in the file result.gal
   for (i=0; i<N; i++) {
       output[i*5 + 0] = particules[i]->pos_x;
       output[i*5 + 1] = particules[i]->pos_y;
       output[i*5 + 2] = particules[i]->mass;
       output[i*5 + 3] = particules[i]->vel_x;
       output[i*5 + 4] = particules[i]->vel_x;
   }

   if (write_doubles_to_file(5*N, output, "result.gal") != 0){
       printf("Error writing file");
       return -1;
   }
```

We use the function "write-doubles_t-file" to write the binary file.

- Extra-part: Graphics

In the argument we can enter 1 if we want a graphic or 0 if not.

```
const float circleRadius = 0.005, circleColor = 0;
const int windowWidth = 800;
float L=1, W=1;
double x, y;

if (graphics == 1){
    InitializeGraphics(argv[0], windowWidth, windowWidth);
    SetCAxes(0,1);
}
```

For the graphic we use circles with 0,005 radius and a square of 800x800 pixels with X and Y axis with L=1 and W=1.

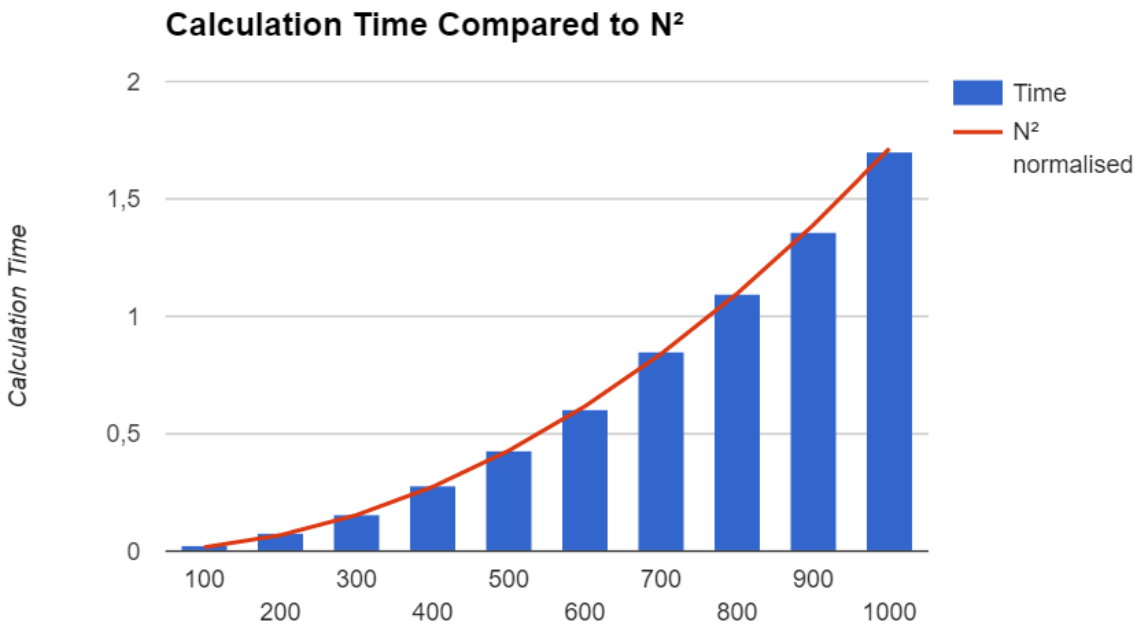
If the argument about graphics is 1 we use the function InitializeGraphics.

```
    for (i=0; i<N; i++){
        particules[i]->pos_x += delta_t*particules[i]->vel_x;
        particules[i]->pos_y += delta_t*particules[i]->vel_y;

        if (graphics == 1){
            x = particules[i]->pos_x;
            y = particules[i]->pos_y;
            DrawCircle(x, y, L, W, circleRadius, circleColor);
        }
    }
    if (graphics == 1){
        Refresh();
        usleep(2000);
    }
}

if (graphics == 1){
    FlushDisplay();
    CloseDisplay();
}
```

For each new position x and y for each particles calculated by sympletic Euler's method the code draw a new circle on the graphic.

Performance

The N-body problem complexity is $O(N^2)$ where N is the number of stars. For performance matters we used both manual and compiler optimizations. After we implemented our solution we made a chart to see if our code's complexity matched the problem's complexity.
The result of that plot is in the figure below:



We can see that the complexities match. To improve the time of calculation in the Euler's method loop we removed the function "pow" of math.h library and computed the power manually since it was integer power. Because calling that function N^2 x nbsteps times in the main loop slowed the program a lot. After doing that the execution time of the program was divided by around 10.

The other improvements in our program were to put outside the loops the affectation of constant value e.g G, G x delta_t. These changes made the program run almost twice faster.

Also, when we compute graphics in the loops, it does not take extra-time because it is a constant value (0|1). For compiler optimizations we used the different optimizers O1, O2, O3 and the -funroll-loops flag. It turned out that unrolling the loops did not have that much effect on the time execution but the best optimizer was O2, so it is the one we used in Makefile.

For a p constant (we calculate the next step):

First we were doing a for loop from 0 to N inside another for loop from 0 to N and each time we calculate the force apply by a particle j on the particle i. But the force apply by i on j is almost the opposite (only the mass used is different). In term of algorithm it allows us to do the half of calculation, the inside loop goes only from i to N. It does not change the complexity but it take the half of the previous time to run. We only had to have 2 arrays that contains the evolution of the sum on both axes and finally update the value of the position and the velocity in another loop.