Parcial 1: Introducción a la computación paralela

1. Simulación de N-Cuerpos

La primera parte del parcial consistía en realizar la implementación de un programa el cual nos permitiera realizar la simulación de N cuerpos celestes con unas condiciones iniciales de posición y velocidad dadas. Siendo así, se realizó la siguiente implementación.

Implementación

Definimos los siguientes `structs` los cuales usaremos para poder facilitar el manejo de los datos, al igual que los principios de encapsulación, a medida que vamos trabajando.

El primero es el `struct` de `vector3d`. Este se usará para poder tener representaciones de vectores de 3 dimensiones de una manera más definida.

```
struct vector3d {
    double x;
    double y;
    double z;
};
```

En cuanto al manejo de las partículas, o en este caso, los cuerpos celestes que se van a estar trabajando, tenemos `space_body` el cual se encarga de tener las características relacionadas con la aceleración, velocidad, entre otras.

```
struct space_body {
    std::string name;
    vector3d position;
    vector3d speed;
    vector3d acceleration;

    double mass;

    void print() {
        printf("%s\n", name.c_str());
        printf("    Position: %e, %e, %e\n",
               position.x,
               position.y,
               position.z);
        printf("    Speed: %e, %e, %e\n",
               speed.x,
               speed.y,
               speed.z);
        printf("    Acceleration: %e, %e, %e\n",
               acceleration.x,
               acceleration.y,
```

```
                acceleration.z);
        printf("    Mass: %e\n", mass);
    }
};
```

Finalmente, como manera de relacionar todos los cuerpos celestes, al igual que todas las funciones las cuales nos permiten calcular las interacciones entre los mismos, tenemos `n_body_system` el cual se encarga de encapsular todas estas.

```
struct n_body_system {
    space_body* bodies;

    double calculate_magnitude(vector3d vec_A, vector3d vec_B) {
        double i = vec_A.x - vec_B.x;
        double j = vec_A.y - vec_B.y;
        double k = vec_A.z - vec_B.z;

        return sqrt(pow(i, 2) + pow(j, 2) + pow(k, 2));
    }

    vector3d calculate_unitary_vector(vector3d vec_A, vector3d vec_B) {
        double magnitude = calculate_magnitude(vec_A, vec_B);

        return vector3d{
            (vec_A.x - vec_B.x) / magnitude,
            (vec_A.y - vec_B.y) / magnitude,
            (vec_A.z - vec_B.z) / magnitude,
        };
    }

    void calculate_accelerations() {
        for (int i = 0; i < TOTAL_BODIES; i++) {
            bodies[i].acceleration.x = 0;
            bodies[i].acceleration.y = 0;
            bodies[i].acceleration.z = 0;

            for (int j = 0; j < TOTAL_BODIES; j++) {
                if (i == j) continue;

                double F = (bodies[j].mass /
                            pow(calculate_magnitude(
                                    bodies[i].position,
                                    bodies[j].position),
                                2));
                vector3d unit_vector = calculate_unitary_vector(
                    bodies[i].position,
                    bodies[j].position);

                bodies[i].acceleration.x += -G * F * unit_vector.x;
                bodies[i].acceleration.y += -G * F * unit_vector.y;
                bodies[i].acceleration.z += -G * F * unit_vector.z;
```

```
            }
        }

        return;
    }

    void update_speeds() {
        for (int i = 0; i < TOTAL_BODIES; i++) {
            bodies[i].speed.x =
                bodies[i].acceleration.x * DELTA_T + bodies[i].speed.x;
            bodies[i].speed.y =
                bodies[i].acceleration.y * DELTA_T + bodies[i].speed.y;
            bodies[i].speed.z =
                bodies[i].acceleration.z * DELTA_T + bodies[i].speed.z;
        }
    }

    void update_positions() {
        for (int i = 0; i < TOTAL_BODIES; i++) {
            bodies[i].position.x =
                bodies[i].speed.x * DELTA_T + bodies[i].position.x;
            bodies[i].position.y =
                bodies[i].speed.y * DELTA_T + bodies[i].position.y;
            bodies[i].position.z =
                bodies[i].speed.z * DELTA_T + bodies[i].position.z;
        }
    }

    void pass_time(int n_delta_ts) {
        for (int i = 0; i < n_delta_ts; i++) {
            calculate_accelerations();
            update_speeds();
            update_positions();
        }
    }

    void print_bodies() {
        for (int i = 0; i < TOTAL_BODIES; i++) {
            bodies[i].print();
        }
    }

    void print_from_center() {
        for (int i = 1; i < TOTAL_BODIES; i++) {
            printf("%s\n", bodies[i].name.c_str());
            printf("    Position: %e, %e, %e\n",
                   bodies[i].position.x - bodies[0].position.x,
                   bodies[i].position.y - bodies[0].position.y,
                   bodies[i].position.z - bodies[0].position.z);
            printf("    Speed: %e, %e, %e\n",
                   bodies[i].speed.x - bodies[0].speed.x,
                   bodies[i].speed.y - bodies[0].speed.y,
                   bodies[i].speed.z - bodies[0].speed.z);
            printf("    Acceleration: %e, %e, %e\n",
```

```
                    bodies[i].acceleration.x,
                    bodies[i].acceleration.y,
                    bodies[i].acceleration.z);
        }
    }
};
```

Ya, a partir de estos 3 `struct`, podemos realizar la simulación de los cuerpos de la siguiente manera:

```
#include <stdio.h>

#include <cmath>
#include <string>

double const G = 6.67e-11;
int const DELTA_T = 86400;
int const N_STEPS = (365 * 10) - 1;
int const TOTAL_BODIES = 5;

int main(int argc, char** argsv) {
    space_body* bodies = new space_body[TOTAL_BODIES];

    bodies[0] = space_body{
        "sun",              // Name
        vector3d{0, 0, 0},  // Position
        vector3d{0, 0, 0},  // Speed
        vector3d{0, 0, 0},  // Acceleration
        1.989e30,           // Mass
    };

    bodies[1] = space_body{
        "mercury",                 // Name
        vector3d{57.909e9, 0, 0},  // Position
        vector3d{0, 47.36e3, 0},   // Speed
        vector3d{0, 0, 0},         // Acceleration
        0.33011e24,                // Mass
    };

    bodies[2] = space_body{
        "venus",                    // Name
        vector3d{108.209e9, 0, 0},  // Position
        vector3d{0, 35.02e3, 0},    // Speed
        vector3d{0, 0, 0},          // Acceleration
        4.8675e24                   // Mass
    };

    bodies[3] = space_body{
        "earth",                    // Name
        vector3d{149.596e9, 0, 0},  // Position
        vector3d{0, 29.78e3, 0},    // Speed
        vector3d{0, 0, 0},          // Acceleration
        5.9724e24                   // Mass
```

```
    };

    bodies[4] = space_body{
        "mars",                     // Name
        vector3d{227.923e9, 0, 0},  // Position
        vector3d{0, 24.07e3, 0},    // Speed
        vector3d{0, 0, 0},          // Acceleration
        0.64171e24                  // Mass
    };

    printf("\nCondiciones Iniciales:\n");
    n_body_system solar_system{
        bodies,
    };

    solar_system.print_bodies();

    solar_system.pass_time(N_STEPS);

    printf("\nCondiciones Finales:\n");
    solar_system.print_bodies();

    return 0;
}
```

Resultados

Tras la ejecución del código propuesto, se imprimirá en la terminal los siguientes resultados:

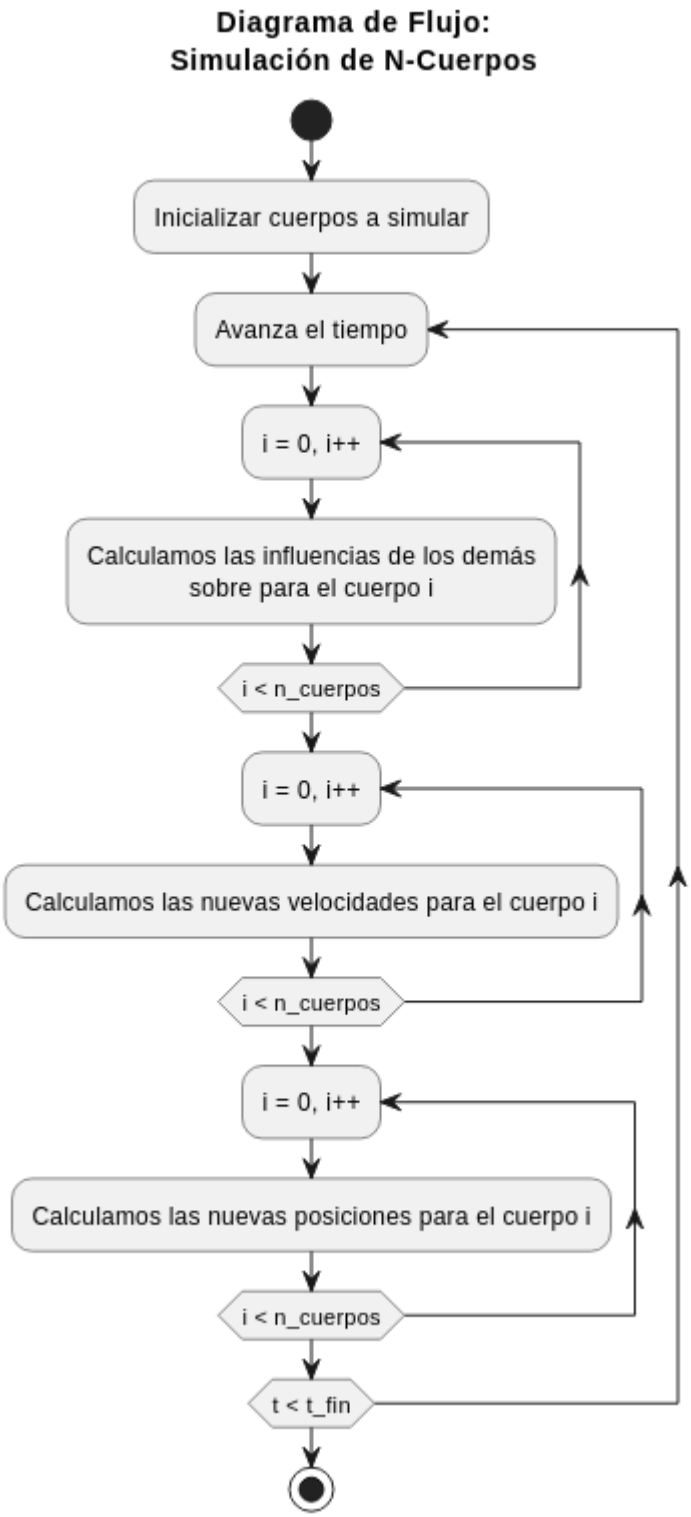
```
Condiciones Iniciales:
sun
    Position: 0.000000e+00, 0.000000e+00, 0.000000e+00
    Speed: 0.000000e+00, 0.000000e+00, 0.000000e+00
    Acceleration: 0.000000e+00, 0.000000e+00, 0.000000e+00
    Mass: 1.989000e+30
mercury
    Position: 5.790900e+10, 0.000000e+00, 0.000000e+00
    Speed: 0.000000e+00, 4.736000e+04, 0.000000e+00
    Acceleration: 0.000000e+00, 0.000000e+00, 0.000000e+00
    Mass: 3.301100e+23
venus
    Position: 1.082090e+11, 0.000000e+00, 0.000000e+00
    Speed: 0.000000e+00, 3.502000e+04, 0.000000e+00
    Acceleration: 0.000000e+00, 0.000000e+00, 0.000000e+00
    Mass: 4.867500e+24
earth
    Position: 1.495960e+11, 0.000000e+00, 0.000000e+00
    Speed: 0.000000e+00, 2.978000e+04, 0.000000e+00
    Acceleration: 0.000000e+00, 0.000000e+00, 0.000000e+00
    Mass: 5.972400e+24
mars
```

```
    Position: 2.279230e+11, 0.000000e+00, 0.000000e+00
    Speed: 0.000000e+00, 2.407000e+04, 0.000000e+00
    Acceleration: 0.000000e+00, 0.000000e+00, 0.000000e+00
    Mass: 6.417100e+23

Condiciones Finales:
sun
    Position: 3.557662e+05, 5.986671e+07, 0.000000e+00
    Speed: 7.832448e-02, 9.630426e-02, 0.000000e+00
    Acceleration: 1.838591e-08, 2.188041e-08, 0.000000e+00
    Mass: 1.989000e+30
mercury
    Position: -3.439939e+10, -4.720540e+10, 0.000000e+00
    Speed: 3.614063e+04, -3.006914e+04, 0.000000e+00
    Acceleration: 2.507497e-02, 2.984974e-02, 0.000000e+00
    Mass: 3.301100e+23
venus
    Position: 1.706067e+10, 1.054625e+11, 0.000000e+00
    Speed: -3.496215e+04, 6.114907e+03, 0.000000e+00
    Acceleration: -2.188175e-03, -1.142790e-02, 0.000000e+00
    Mass: 4.867500e+24
earth
    Position: 1.491491e+11, -1.270979e+10, 0.000000e+00
    Speed: 2.539061e+03, 2.965212e+04, 0.000000e+00
    Acceleration: -5.887531e-03, 6.061004e-04, 0.000000e+00
    Mass: 5.972400e+24
mars
    Position: -1.317618e+11, 1.829618e+11, 0.000000e+00
    Speed: -1.979680e+04, -1.415560e+04, 0.000000e+00
    Acceleration: 1.506214e-03, -2.132460e-03, 0.000000e+00
    Mass: 6.417100e+23
```

2. Oportunidades de Paralelización

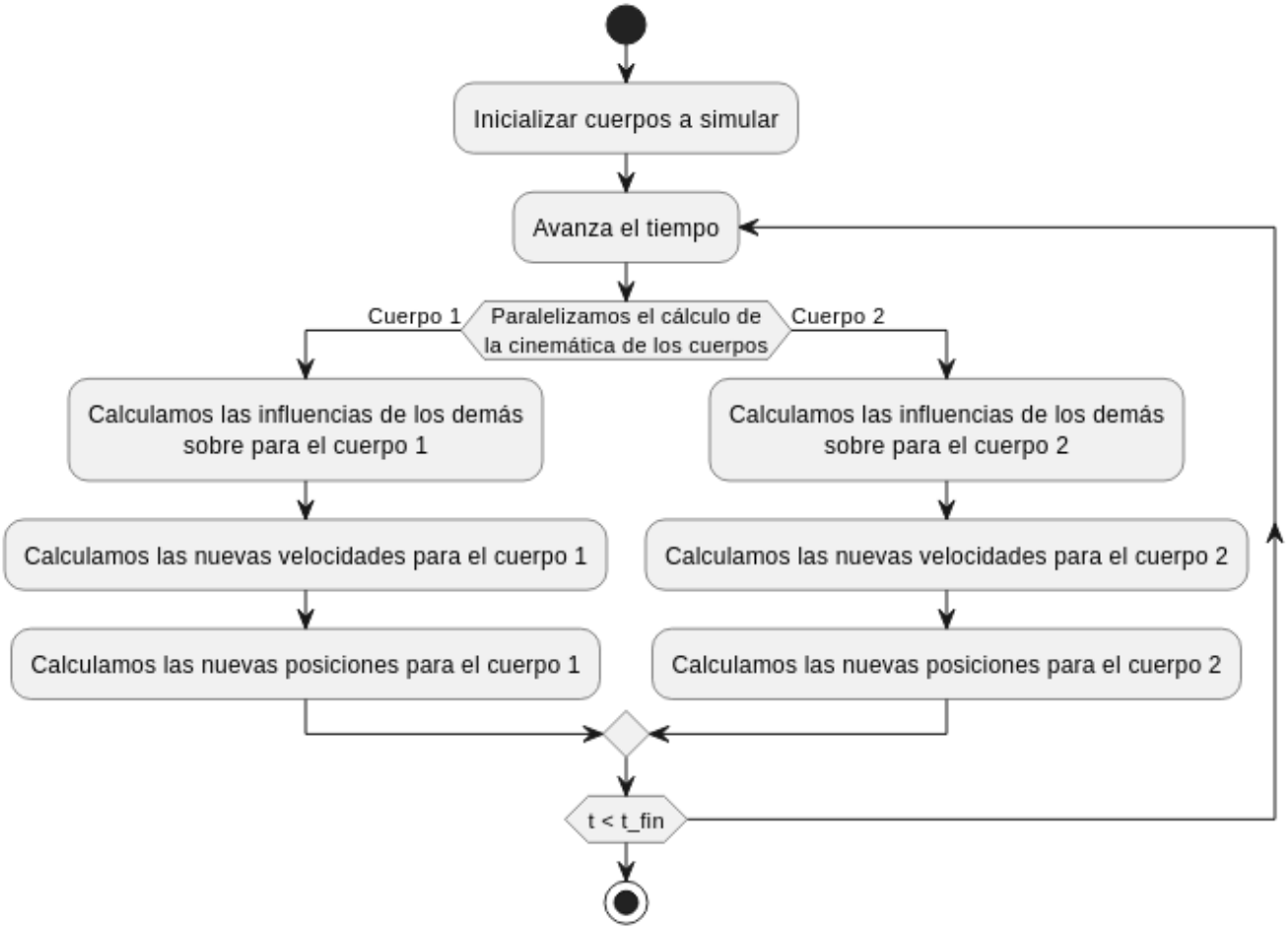
Viendo la implementación del problema, se puede identificar una serie de pasos, los cuales tienen que seguirse para poder calcular la aceleración, velocidad y la posición de cada uno de los cuerpos dentro del sistema. Estos pasos pueden verse en el siguiente diagrama:



Como podemos ver, en el caso serial, calculamos las aceleraciones que dependen de las posiciones de todos los cuerpos en el sistema y luego los cambios en su velocidad y posición. Entonces, partiendo de que la aceleración depende únicamente de las posiciones de los cuerpos en el instante de tiempo anterior, podemos realizar el cálculo de las aceleraciones de todos los cuerpos de manera independiente. Siendo así, podemos paralelizar este proceso.

Así mismo, con la nueva aceleración, y los valores de posición y velocidad anteriores de nuestro cuerpo a trabajar, podemos calcular los nuevos valores de velocidad y posición. Es decir, podemos calcular los cambios en la aceleración, posición y velocidad de manera independiente sin afectar a los demás cuerpos dentro de un mismo instante de tiempo. Veamos como se vería en el caso de paralelizar 2 cuerpos:

Diagrama Explicativo: Paralelización



Expandiendo sobre esta ideqa, podemos realizar la paralelización de n cuerpos dependiendo de la cantidad de ranks que tengamos disponibles, y al no presentarse un desbalance en el caso de calcular un cuerpo u otro, podemos simplemente repartir en cantidades iguales los cuerpos que estemos calculando para cada uno de los nodos disponibles en nuestro cluster.

En cuanto a la comunicación y la aglomeración entre los nodos, se esperaría que tras mandar los cuerpos a calcular, cada uno de los nodos reporte las nuevas condiciones al nodo "orquestador" el cual esperaría las respuestas de todos los ranks usados, antes de calcular el siguiente instante de tiempo.

Cabe resaltar que esta puede que no sea la mejor propuesta de paralelización debido a que sólo está optimizando una pequeña parte del proceso y no el *bulk* de la operación que es la parte iterativa. Sin embargo, para casos con una gran cantidad de cuerpos, esta parelización podría reducir en gran medida el tiempo de ejecución del problema.

Daniel David Delgado Cervantes - 2182066