Práctica de programación orientada al rendimiento

J. Daniel García Sánchez (coordinador)

Arquitectura de Computadores
Departamento de Informática
Universidad Carlos III de Madrid

1. Objetivo

Este proyecto tiene como objetivo fundamental hacer que los estudiantes adquieran **preocupación por el rendimiento de programas secuenciales** y se familiaricen con la **optimización de los mismos**. Así mismo, se familiarizarán con las **técnicas de evaluación del rendimiento**.

En concreto, la práctica se centrará en el desarrollo de software secuencial en el lenguaje de programación C++ (incluyendo sus últimas versiones).

2. Descripción del proyecto

Para la realización de este proyecto se debe implementar en C++ una aplicación de simulación gravitatoria de objetos. El programa debe utilizar únicamente técnicas de programación secuencial y no hará uso del concurrencia ni paralelismo.

Se implementará dos versiones del programa utilizando las estrategias: arrays de estructuras (**aos**) y estructuras de arrays (**soa**).

Para una correcta realización del proyecto se suministran, en las siguientes secciones de este enunciado, los requisitos que deben cumplir los programas desarrollados, así como un archivo binario con una versión del programa implementado para facilitar la comparación de los resultados.

2.1. El problema de la atracción gravitatoria

El problema que se debe resolver consiste en una simulación a lo largo del tiempo de las posiciones de un conjunto de objetos en un recinto. Este cálculo se realiza a lo largo de la ejecución del programa a intervalos regulares de tiempo de longitud Δt.

El programa debe realizar la simulación en un espacio tridimensional. Se considera que el espacio es un recinto cerrado. De esta manera cuando un objeto impacta con el borde del recinto, se produce un rebote, cambiando la dirección del movimiento del objeto.

Finalmente, en cada incremento de tiempo Δt se verifican las posibles colisiones que se producen entre los distintos objetos.

De esta forma, se pretende que los estudiantes simulen el comportamiento de N objetos moviéndose en el espacio tridimensional.

2.2. Requisitos

2.2.1. Ejecución del programa y parámetros de entrada

- Se deberá desarrollar dos programas llamados **sim-soa** (versión de estructuras de arrays) y **sim-aos** (versión arrays de estructuras).
- Los parámetros que deberán recibir los programas para su correcta ejecución son los siguientes:
  - **num_objects**: es un número entero, mayor que 0, que indica el número de objetos que se van a simular.
  - **num_iterations**: es el número entero, mayor que 0, que indica el número de iteraciones (pasos de tiempo) que se van a simular.
  - **random_seed**: es un número entero positivo que sirve como semilla para las funciones generadoras de distribuciones aleatorias.
    Nota: Dos simulaciones con los mismos parámetros pero distinta semilla darán lugar a escenarios distintos.
  - **size_enclosure**: es un número real positivo que indica el tamaño del recinto de simulación. El recinto se considera un cubo perfecto con vértice en el origen de coordenadas y con lado el valor **size_enclosure**.
    De este modo, si el valor para **size_enclosure** es n, los vértices opuestos del cubo tendrán coordenadas $(0, 0, 0)$ y (n, n, n).
  - **time_step**: es un número real positivo que indica el incremento de tiempo en cada iteración de la simulación.
- En el caso de que el número de parámetros sea incorrecto, se procederá a terminar el programa con código de error -1 y un mensaje de error por la salida de errores estándar.

```
user@maquina:~$
user@maquina:~$./sim-soa 100 2000 1
sim-soa invoked with 3 parameters.
Arguments:
  num_objects: 100
  num_iterations: 2000
  random_seed: 1
  size_enclosure: ?
  time_step: ?
```

- En el caso de que alguno de los parámetros no sean correctos se procederá a terminar el programa con código de error -2 y un mensaje de error por la salida estándar de errores.

```
user@maquina:~$
user@maquina:~$./sim-soa 100 2000 1 -1000000 0.1
Error: Invalid box size
sim-soa invoked with 5 parameters.
Arguments:
  num_objects: 100
  num_iterations: 2000
  random_seed: 1
  size_enclosure: -1000000
  time_step: 0.1
```

- Una vez que se hayan procesado todos los parámetros de entrada y se hayan calculado las posiciones y velocidades iniciales de los elementos de la simulación, se debe escribir esta información en un fichero con el nombre **init_config.txt**, con el siguiente formato:

- Todos los valores reales se imprimirán con tres decimales en la parte fraccionaria.
- Una línea de cabecera que incluirá:
  - Tamaño del recinto.
  - Paso de tiempo.
  - Número de objetos iniciales.
- Una línea por objeto en la que se debe escribir su posición (coordenadas x, y, z), su velocidad (componentes x, y, z) y su masa. Estos siete valores aparecerán separados por espacios.
- Ejemplo para un recinto de tamaño 1000000,0, con paso de tiempo 0,100 y 2000 objetos iniciales.

```
1000000.000 0.100 2000
903604.026 850236.140 783820.465 0.000 0.000 0.000 99999999999999868928.000
135885.825 224540.656 99650.335 0.000 0.000 0.000 100000000000000262144.000
22087.739 685842.870 654084.972 0.000 0.000 0.000 99999999999999737856.000
...
```

Nota: Esto es un ejemplo del formato de fichero. Los datos no se corresponden necesariament con los obtenidos con el programa y los parámetros de entrada dados en el ejemplo.

2.2.2. Generación de los parámetros de simulación

- La posición de los objetos en el espacio tridimensional (con coordenadas en coma flotante de doble precisión) y su masa, se deben generar usando distribuciones aleatorias del siguiente modo:
  - Se debe utilizar una única instancia de un motor de generación de números pseudoaletarios para todas las distribuciones.
  - El generador de números será un Mersenne-Twister de 64 bits (**std::mt19937_64**). Este motor se iniciará con la semilla aleatoria pasada como argumento al programa. (Puede consultar información sobre este tipo en https://en.cppreference.com/w/cpp/numeric/random/mersenne_twister_engine).
  - Todos los valores de posiciones se generarán utilizando una distribución uniforme con límite inferior **0** y límite superior **tam_recinto**, utilizando un objeto del tipo **uniform_real_distribution**. (Puede consultar información en https://en.cppreference.com/w/cpp/numeric/random/uniform_real_distribution).
  - Todos los valores de masas se generarán utilizando una distribución normal con media 10^{21} y desviación típica 10^{15} utilizando un objeto del tipo **normal_distribution**. (Puedes consultar información en https://en.cppreference.com/w/cpp/numeric/random/normal_distribution).
- El orden de generación de los valores es importante. Se deben generar siempre objeto a objeto y dentro de cada objeto siguiendo el orden $< x, y, z, m >$
  - x_0, y_0, z_0, m_0.
  - x_1, y_1, z_1, m_1
  - x_2, y_2, z_2, m_2
  - ...

2.2.3. Cálculo de las fuerzas de atracción

Nota: se recomienda seguir el orden de las operaciones que se describen a continuación, ya que un orden distinto puede producir resultados inexactos.

- **Cálculo de fuerza gravitatoria**: Se usará la constante de gravitación (G), las masas de los dos objetos (m_i y m_j), y los vectores de posición de ambos objetos ($\vec{p_i}$ y $\vec{p_j}$).

$$\vec{F_{ij}} = \frac{G \cdot m_i \cdot m_j}{||\vec{p_j} - \vec{p_i}||^2} \cdot \frac{(\vec{p_j} - \vec{p_i})}{||\vec{p_j} - \vec{p_i}||} = \frac{G \cdot m_i \cdot m_j \cdot (\vec{p_j} - \vec{p_i})}{||\vec{p_j} - \vec{p_i}||^3}$$

  Donde la norma sobre un vector $\vec{v}$, es la **norma euclídea**.

  Se tomará para la constante de gravitación universal el valor $6{,}674 \cdot 10^{-11}$.

  Para el objeto **i**, esta fuerza se aplicará de forma positiva y para el objeto **j** de forma negativa. Un elemento no ejercerá fuerza sobre sí mismo.

$$F_{ij} = -F_{ji}$$

$$F_{ii} = 0$$

- **Cálculo del vector aceleración**: La aceleración ($veca$) se determinará a partir de la masa del objeto (m) y de las contribuciones de todas las fuerzas

$$\vec{a_i} = \frac{1}{m_i} \sum_{j=1}^{N} \vec{F_{ij}}$$

- **Cálculo del vector velocidad**: La velocidad ($\vec{v}$) se determinara a partir de la aceleración ($\vec{a}$) y del incremento de tiempo (Δt):

$$\vec{v_i} = \vec{v_i} + \vec{a_i} \cdot \Delta t$$

- **Cálculo del vector posición**: La posición se determinará a partir de la velocidad ($\vec{v}$) y del incremento de tiempo (Δt):

$$\vec{p_i} = \vec{p_i} + \vec{v_i} \cdot \Delta t$$

- **Efecto rebote**: Si la posición traspasa uno de los límites del recinto:

  1. El objeto se coloca en el correspondiente plano límite:

$$p_x \leq 0 \Rightarrow p_x \leftarrow 0$$

$$p_y \leq 0 \Rightarrow p_y \leftarrow 0$$

$$p_z \leq 0 \Rightarrow p_z \leftarrow 0$$

$$p_x \geq tam \Rightarrow p_x \leftarrow tam$$

$$p_y \geq tam \Rightarrow p_y \leftarrow tam$$

$$p_z \geq tam \Rightarrow p_z \leftarrow tam$$

  2. Además su componente del vector velocidad en el correspondiente eje deberá cambiarse de signo.

2.2.4. Colisiones entre objetos

Cuando dos objetos colisionan se colapsan en un objeto de mayor masa. Se considera que dos objetos han colisionado si su distancia es menor que la unidad.

En este caso los dos objetos (a y b), dan lugar a un solo objeto (c) con las siguientes propiedades:

- La masa es la suma de las masas ($m_c = m_a + m_b$).
- La velocidad es la suma de las velocidades ($\vec{v}_c = \vec{v}_a + \vec{v}_b$).
- La posición es la del primer objeto ($\vec{p}_c = \vec{p}_a$).
- El objeto a es remplazado por el objeto c.
- El objeto b deja de existir.

Nota: Tenga en cuenta que en estos casos se pasa a tener un objeto menos en el sistema.

IMPORTANTE: Debe comprobarse que dos objetos no colisionan al final de cada iteración de simulación. También debe realizarse esta comprobación antes de iniciar la simulación.

2.2.5. Almacenamiento de los datos

Una vez finalizados todas las iteraciones de la simulación se debe realizar un almacenamiento de los datos finales de la misma, en un archivo llamado **final_config.txt**, que tendrá el mismo formato y contenidos que el archivo **init_config.txt**.

3. Tareas

3.1. Desarrollo de versión secuencial

Esta tarea consiste en el desarrollo de la versión secuencial de la aplicación descrita en C++17 (o C++20).

3.1.1. Configuración del compilador

Todos sus archivos fuente deben compilar sin problemas y no deben emitir ninguna advertencia del compilador. En particular, deberá activar, al menos, los siguientes flags del compilador relativos a *warnings*:

- `-Wall -Wextra -Wno-deprecated -Werror -pedantic -pedantic-errors`,

Tenga también en cuenta que deberá realizar todas las evaluaciones con las optimizaciones del compilador activadas (opciones del compilador `-O3` y `-DNDEBUG`). Puede conseguir esto de una forma sencilla con `-DCMAKE_BUILD_TYPE=Release`.

Se podrán activar flags adicionales de optimización siempre que se documente en la memoria del proyecto y se justifique la utilización de los mismos. Debe incluir dichos flags en su archivo de configuración de CMake.

3.1.2. Bibliotecas

Está permitido el uso de cualquier componente de la biblioteca estándar de C++ que está incluida en la distribución del compilador. Esto incluye muchos archivos de cabecera como **<vector>**, **<list>**, **<fstream>**, **<cmath>**, **<random>**, ...

En general, no se permiten bibliotecas adicionales. No obstante, el coordinador de la asignatura podrá conceder permisos a aquellos estudiantes que lo soliciten de manera justificada.

3.2. Evaluación del rendimiento secuencial

Esta tarea consiste en evaluar el rendimiento de la aplicación secuencial.

Para evaluar el rendimiento debe medir el tiempo de ejecución de la aplicación. Debe representar gráficamente los resultados. Tenga en cuenta las siguientes consideraciones:

- Todas las evaluaciones deben realizarse en las aulas de la universidad. Debe incluir en la memoria todos los parámetros relevantes de la máquina en la que ha ejecutado (modelo de procesador, número de cores, tamaño de memoria principal, jerarquía de la memoria caché, ...) y del software de sistema (versión de sistema operativo, versión del compilador, ...).
    - Alternativamente, los estudiantes podrán realizar las evaluaciones en su propio computador siempre que este disponga de características similares. En caso de duda, los estudiantes deben dirigirse al profesor responsable del grupo reducido en el que están matriculados.
- Realice cada experimento un número de veces y tome el valor promedio. Se recomienda un mínimo de 10 o más ejecuciones por experimento y que proporcione un intervalo de confianza.
- Estudie los resultados para varios tamaños de la población de objetos. Considere los casos de 1000, 2000 y 4000.
- Estudie los resultados para distintos números de iteraciones: 50, 100 y 200.

Represente en una gráfica todos los tiempos totales de ejecución obtenidos. Represente en otra gráfica el tiempo medio por iteración.

Incluya en la memoria de esta práctica las conclusiones que pueda inferir de los resultados. No se limite simplemente a describir los datos. Debe buscar también una explicación convincente de los resultados.

4. Calificación

La puntuación final obtenida en este proyecto se obtiene teniendo en cuenta el siguiente reparto:

- **Rendimiento alcanzado** (50 %).
- **Pruebas funcionales** (20 %).
- **Calidad de la memoria** (30 %).

Advertencias:

- Si el código entregado no compila, la nota final de la práctica será de 0.
- En caso de copia todos los grupos implicados obtendrán una nota de 0.

5. Procedimiento de entrega

La entrega del código y de la correspondiente memoria del proyecto se realizará a través de Aula Global.

- **Entrega del código fuente**

- Se entregará un archivo comprimido en formato **zip** que contendrá todo el código fuente de la aplicación.
- Deben incluirse todos los archivos **.hpp** y **.cpp** utilizados.
- También deben incluirse todos los archivos CMake para generar el proceso de compilación.
- No debe incluirse ningún archivo resultado de la compilación como archivos objeto o ejecutables.

- **Memoria del proyecto**

  - Debe entregarse una memoria en formato PDF.
  - Evite incluir en la memoria código fuente. Si fuese necesaria haga referencia al código fuente entregado.
  - La memoria deberá incluir los siguientes contenidos:
    - **Página de título**. Contendrá:
      - El nombre de la práctica.
      - El grupo reducido en el que están matriculados los estudiantes.
      - El número de equipo asignado.
      - El nombre y NIA de los autores.
  - **Índice de contenidos**.
  - **Diseño original**. Explicará el diseño general de la aplicación especificando la estructura general, las decisiones de diseño adoptadas, las alternativas consideradas y los principales tipos de datos utilizados.
  - **Optimización**. Contendrá una discusión de las optimizaciones aplicadas y su impacto. Opcionalmente, podrá incluir (si fuese el caso) la utilización de flags adicionales de optimización explicando sus ventajas e inconvenientes.
  - **Evaluación de rendimiento**. Presentará las diversas evaluaciones del rendimiento realizadas comparando la versión original desarrollada y la versión optimizada.
  - **Pruebas realizadas**. Descripción del plan de pruebas realizadas para asegurar la correcta ejecución de ambas versiones. Deberá contener pruebas funcionales y, si fuese necesario, pruebas unitarias.
  - **Conclusiones**. Se valorará especialmente las derivadas de los resultados de la evaluación del rendimiento, así como las que relacionen el trabajo realizado con el contenido de la asignatura.

La memoria no deberá sobrepasar las 15 páginas incluyendo la portada y todas las secciones. No se tendrá en cuenta en la corrección los contenidos a partir de la página 16 si fuese el caso.