Práctica de programación paralela con OpenMP

J. Daniel García Sánchez (coordinador)
Alberto Cascajo
Diego Camarmas
Elías del Pozo

Arquitectura de Computadores
Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid

1. Objetivo

Esta práctica tiene como objetivo fundamental hacer el que los estudiantes se familiaricen con la **optimización de programas secuenciales** y con los **modelos de programación paralela** en arquitecturas de memoria compartida.

En concreto, la práctica se centrará en el desarrollo de software secuencial en el lenguaje de programación C++ (incluyendo las mejoras de C++14), así como en las técnicas de paralelismo mediante el uso de *OpenMP*.

2. Descripción de la práctica

Para la realización de esta práctica se debe implementar en C++ dos versiones de una variante del problema de los asteroides, una basada en el modelo de programación secuencial, y otra basada en el modelo paralelo.

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

2.1. El problema de los asteroides

El problema que se debe resolver consiste en una simulación a lo largo del tiempo del movimiento de un conjunto de asteroides teniendo en cuenta la atracción gravitatoria que producen unos sobre otros. 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 bidimensional. 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.

Adicionalmente, en los bordes del espacio existen otros cuerpos fijos (planetas), los cuáles ejercen fuerzas de atracción gravitatoria sobre los asteroides, pero no se ven afectados por las que se ejercen sobre ellos (son fijos).

Finalmente, en cada incremento de tiempo Δt se verifican los posibles rebotes que se producen entre los propios asteroides y éstos con los bordes del espacio.

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

2.2. Requisitos

2.2.1. Ejecución del programa y parámetros de entrada

- Se deberá desarrollar dos versiones distintas del programa: una versión secuencial y otra versión paralela:
  - **nasteroids-seq**: Versión secuencial.
  - **nasteroids-par**: Versión paralela.
- Los parámetros que deberá recibir el programa para su correcta ejecución son los siguientes:
  - **num_asteroides**: es un número entero, mayor o igual que 0, que indica el número de asteroides que se van a simular.
  - **num_iteraciones**: es el número entero, mayor o igual que 0, que indica el número de iteraciones (pasos de tiempo) que se van a simular.
  - **num_planetas** es un número entero, mayor o igual que 0, que indica el número de planetas que se van a incluir en los bordes del espacio.
  - **semilla** es un número entero positivo que sirve como semilla para las funciones generadoras de número aleatorios.
  - **Nota**: Dos simulaciones con los mismos parámetros pero distinta semilla darán lugar a escenarios distintos.
- En el caso de que alguno de los parámetros no sean correctos o no estén presentes se procederá a terminar el programa con código de error -1 y un mensaje de error.
  - En el caso de **nasteroids-seq**:

```
nasteroids-seq: Wrong arguments.
Correct use:
nasteroids-seq num_asteroides num_iteraciones num_planetas semilla
```

  - En el caso de **nasteroids-par**:

```
nasteroids-par: Wrong arguments.
Correct use:
nasteroids-par num_asteroides num_iteraciones num_planetas semilla
```

- Una vez que se hayan procesado todos los parámetros de entrada y se hayan calculado las posiciones iniciales de los elementos de la simulación, se debe escribir esta información en un fichero con el nombre **init_conf.txt**, con un dato en cada línea:
  - Parámetros de entrada separados por un espacio en el mismo orden en el que se introdujeron (sin incluir el nombre del programa).
  - Para cada asteroide primero y planeta después se debe almacenar su posición inicial (primero x, después y) y su masa. Cada elemento en una línea con sus parámetros separados por espacios. En el caso de imprimir números en coma flotante, se deberán imprimir 3 decimales.

- Ejemplo:

```
5 2 1 2000
2.567 2.573 18.000
3.545 2.523 14.233
45.545 9.000 15.644
2.567 2.556 19.555
89.965 2.335 18.577
0.000 15.700 180.577
```

Ejemplo para 5 asteroides, 2 iteraciones, 1 planeta, y 2000 como semilla.

Nota: Esto es un ejemplo del formato de fichero. Los datos en el contenido no se tienen por qué corresponder con aquéllos 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 asteroides en el espacio bidimensional (dos coordenadas en coma flotante de doble precisión), se debe generar en base a una distribución aleatoria con la semilla que se ha indicado por línea de comandos. Para ello se debe hacer uso de las siguientes instrucciones:

```
// Random distributions
default_random_engine re{seed};
uniform_real_distribution<double> xdist{0.0, std::nextafter(width,
                                  std::numeric_limits<double>::max())};
uniform_real_distribution<double> ydist{0.0, std::nextafter(height,
                                  std::numeric_limits<double>::max())};
normal_distribution<double> mdist{mass, sdm};
```

Listing 1: Códificación de las distribuciones aleatorias.

- Para asignar un valor a la coordenada "x" de un asteroide (por ejemplo), la posición se puede obtener mediante la evaluación de la expresión **xdist(re)**.
- Los planetas se deben situar en los bordes del espacio generado, comenzando en el eje izquierdo ($x = 0$), el segundo en el superior ($y = 0$), el tercero en el borde derecho, el cuarto en el inferior, etc. La segunda coordenada seguirá, al igual que los asteroides, una distribución aleatoria con la misma semilla. En cuanto a su masa, se calcula igual que la de los asteroides, pero multiplicada por un valor constante de 10.

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.

- **Distancia**: Para cada asteroide se debe calcular la contribución de los demás elementos (asteroides y planetas) a su movimiento:

  1. Calcular distancia entre asteroide y elemento:

$$dist(a, b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

- **Movimiento normal**: Si la distancia es mayor que 5 (constante de mínimo de distancia):

  1. Calcular ángulo de influencia:

   a) Cálculo de la **pendiente**:

$$pendiente = \frac{y_a - y_b}{x_a - x_b}$$

   b) Corrección de la pendiente:
      - Si la pendinete es mayor que 1, ésta se fijará a 1.
      - Si la pendiente es menor que -1, ésta se fijará a -1.

   c) Cálculo del **ángulo**: Se determina mediante la arcotangente de la pendiente.

$$\alpha = atan(pendiente)$$

2. Calcular **fuerza de atracción**: Se usará la constante de gravitación (G), las masas de los dos asteroides (m_a y m_b), la distancia entre ambos ($dist(a, b)$) y el ángulo determinado anteriormente (α). El valor máximo de la fuerza de atracción será de 100, por lo que cualquier cálculo superior deberá ser truncado a este valor:

$$f_x = \frac{G \cdot m_a \cdot m_b}{dist(a,b)^2} \cdot cos(\alpha)$$

$$f_y = \frac{G * m_a \cdot m_b}{dist(a,b)^2} \cdot sin(\alpha)$$

   Para el asteroide **a**, esta fuerza se aplicará de forma positiva y para el asteroide **b** de forma negativa. Un elemento no ejercerá fuerza sobre sí mismo.

   Nota: La fuerza que **a** ejerce sobre **b** tiene el mismo valor que la que **b** ejerce sobre **a** pero con sentido contrario.

3. Para cada fuerza calculada y ángulo, aplicar a su asteroide:

   a) Cálculo de la aceleración (a), a partir de la masa (m) y la fuerza (f) correspondiente en cada eje:

$$a = \frac{1}{m} \sum f$$

   b) Cálculo de la velocidad (v) para cada eje (x e y):

$$v = v + a \cdot \Delta t$$

   c) Modificación de la posición (p) en x e y, debida a la velocidad de cada eje:

$$p = p + v \cdot \Delta t$$

- **Efecto rebote**: si el asteroide llega a un borde del espacio:

1. Deberá posicionarse a 5 puntos del límite del espacio:

$$x \leq 0 \Rightarrow x \leftarrow 5$$

$$y \leq 0 \Rightarrow y \leftarrow 5$$

$$x \geq width \Rightarrow x \leftarrow width - 5$$

$$y \geq height \Rightarrow y \leftarrow height - 5$$

2. Además su componente de velocidad en el eje de choque deberá multiplicarse por -1 ($v_x = -v_x$ o $v_y = -v_y$).

2.2.4. Rebote entre asteroides

Como se ha podido leer, el cálculo de las fuerzas entre asteroides se realiza si éstos están a una distancia mínima de 5. Para los casos en los que la distancia es menor o igual que dicho valor se debe evaluar el rebote entre asteroides.

La resolución de este caso se basa en un intercambio en los componentes de la velocidad de los asteroides, ya que, cuando dos elementos **a** y **b** en movimiento chocan, el resultado puede parecerse a una continuación de la trayectoria del elemento contrario.

$$V_x a = V_x b$$

$$V_y a = V_y b$$

$$V_x b = V_x a$$

$$V_y b = V_y a$$

2.2.5. Constantes a utilizar

- Constante de gravitación universal en un universo paralelo: $gravity = 6{,}674e-5$.
- Intervalo de tiempo: $\Delta t = 0{,}1$.
- Distancia mínima: $dmin = 5{,}0$.
- Anchura del espacio: $width = 200$.
- Altura del espacio: $height = 200$.
- Media para el cálculo de la distribución normal de las masas: $m = 1000$.
- Desviación estándar para el cálculo de la dist. normal de las masas $sdm = 50$.

2.2.6. 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 fichero llamado **out.txt**, que incluyen los siguientes datos:
  - Posiciones finales de los asteroides (x, y).
  - Sus velocidades (x, y).
  - Sus masas.

  Los datos de cada asteroide se colocarán en líneas independientes con sus parámetros separados por espacios. En el caso de imprimir números en coma flotante, se deberán imprimir 3 decimales.

- Ejemplo:

```
2.567 2.573 45.567 28.573 18.000
8.567 102.573 4.567 67.573 19.760
```

  Ejemplo para una simulación en la que se han lanzado dos asteroides.

  Nota: Esto es un ejemplo del formato de fichero. Los datos en el contenido no se tienen por qué corresponder con aquéllos obtenidos con el programa y los parámetros de entrada dados en el ejemplo.

- Dada la complejidad de que dos programadores que trabajan con números en coma flotante obtengan los mismos resultados ante un mismo problema, se proporciona una funcionalidad en el binario de ejemplo (ejecutable *step_by_step* descarbale desde Aula Global) que imprime en un fichero llamado *stepbystep.txt* los datos intermedios al final de cada iteración. Este programa no tiene que ser implementado por los estudaintes, sino que es una ayuda que pueden usar para validar el trabajo realizado. Cada línea indica la interacción entre dos elementos, su fuerza calculada y su ángulo. El objetivo es que se puedan comprobar los datos entre la aplicación base y el programa generado por los estudiantes. Dicho fichero tiene el siguiente formato:

```
******* ITERATION ******
--- asteroids vs asteroids ---
aster_i aster_j fuerza_calculada angulo
...
--- asteroids vs planets ---
planet_i aster_j fuerza_calculada angulo
...
```

  Donde *aster_i* representa la posición del elemento en la estructura de almacenamiento (primer asteroide, segundo, ..., por orden de creación).

3. Paralelismo con OpenMP

Para la versión paralela los requisitos del programa son idénticos, por lo que los **resultados numéricos deben ser los mismos que en la versión secuencial**. La principal diferencia es la utilización de todos los núcleos disponibles en el sistema multicore utilizado a través de la aplicación del modelo de programación paralela basado en OpenMP.

Nota: para la comprobación de los resultados, los ficheros generados por el estudiante deben ser iguales que los generados por el binario proporcionado por los profesores, con los mismos valores de entrada. Para la verificación de este apartado se recomienda la utilización del mandato "diff" de linux con los archivos a comparar.

4. Tareas

4.1. Desarrollo de versión secuencial

Esta tarea consiste en el desarrollo de la versión secuencial de la aplicación descrita en C++14.

Todos sus archivos fuente deben compilar sin problemas y no deben emitir ninguna advertencia del compilador. En particular deberá activar los siguientes flags del compilador:

- `-std=c++14 -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`).

Se podrá utilizar la librería matemática estándar de C++ y las flags necesarias para su enlazado.

4.2. 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 un procesador de al menos 4 núcleos físicos (excluyendo *hyperthreading*). En este caso el sistema operativo deberá ser Linux.
  - Tenga en cuenta que la evaluación de la parte secuencial y la parte paralela deberá realizarse en el mismo computador y con las mismas condiciones de entorno.
- Realice cada experimento un número de veces y tome el valor promedio. Se recomienda un mínimo de 10 ejecuciones por experimento.
- Estudie los resultados para varios tamaños de la población de objetos. Considere los casos de 250, 500 y 1000.
- 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.

4.3. Paralelización

Esta tarea consiste en desarrollar la correspondiente versión paralela de los programas suministrados usando OpenMP. Deberá explicar en la memoria de forma detallada las modificaciones que ha realizado al código.

Tenga en cuenta lo siguiente:

- Cualquier práctica que emita un advertencia (*warning*) se considerará suspensa.
- Sus programas deben incluir entre los parámetros pasados al compilador, como mínimo, los siguientes: `-std=c++14 -Wall -Wextra -Wno-deprecated -Werror -pedantic -pedantic-errors`, o parámetros equivalentes.

En la memoria debe incluir las decisiones de diseño que ha tomada para alcanzar la paralelización. Para cada decisión debe incluir que otras alternativa se podían considerar y justificar de forma razonada los motivos de la elección realizada.

Se podrá utilizar la librería matemática estándar de C++ y las flags necesarias para su enlazado.

4.4. Evaluación de versión paralela

Debe repetir las evaluaciones del primer apartado. Considere distinto número de hilos de ejecución, variando desde 1 hasta 16 hilos (1, 2, 4, 8 y 16).

Nota importante: Asegúrese de compilar todos los ejemplos con las optimizaciones activadas (opciones del compilador `-O3` y `-DNDEBUG`).

Además de las gráficas del primer apartado, represente de forma gráfica el *speedup*.

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 que incluya el impacto de la arquitectura del computador.

4.5. Impacto de la planificación

Realice un estudio para los casos de 4 y de 8 hilos del impacto que tienen los distintos modelos de planificación (*static*, *dynamic* y *guided*) incluidos en OpenMP. Incluya en la memoria las conclusiones que pueda inferir de los resultados obtenidos.

5. Calificación

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

- Calificación de la implementación: 100 %
  - Implementación secuencial 40 %
    - Pruebas funcionales 30 %
    - Rendimiento alcanzado 30 %
    - Memoria 40 %
  - Implementación paralela 60 %
    - Pruebas funcionales 30 %
    - Rendimiento alcanzado 30 %
    - Memoria 40 %

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.

6. Procedimiento de entrega

La entrega del código se realizará a través de Aula Global:

- El código final junto con la memoria se deberá subir al correspondiente entregador de Aula Global.
- Deberá estar comprimido en un .zip con el nombre **popenmp2019.zip**.
  - Un archivo **autores.txt** que contendrá:
    - Una línea con el número del grupo en el que están matriculados los integrantes del grupo.
    - Una línea por cada integrante del grupo con el NIA, apellidos y nombre.
  - **Carpeta con el código secuencial**: deberá tener nombre **seq** y **contener todos los archivos necesarios para la compilación y ejecución del código**.
  - **Carpeta con el código paralelo**: deberá tener nombre **par** y **contener todos los archivos necesarios para la compilación y ejecución del código**.

Se habilitará un entregador separado para la entrega de la memoria. La misma deberá contener al menos las siguientes secciones:

- Página de título: contendrá el nombre de la práctica, el nombre y NIA de los autores y el número de grupo pequeño.
- Índice de contenidos.
- Introducción.
- Versión secuencial: Sección en la que se explicará la implementación y optimización del código secuencial. No incluir código en ella.
- Versión paralela: Sección en la que se explicará la implementación y optimización del código secuencial. No incluir código en ella.
- Evaluación de rendimiento: Sección en la que se explicarán las diversas evaluaciones realizadas comparando el programa inicial, la versión secuencial del alumno y la versión paralela optimizada por el alumno.
- Pruebas realizadas: Descripción del plan de pruebas realizadas para asegurar la correcta ejecución de ambas versiones.
- Conclusiones.

La memoria no deberá sobrepasar las 15 páginas incluyendo todas las secciones y deberá estar en formato PDF.