UNIVERSITAT DE LLEIDA

ESCOLA POLITÈCNICA SUPERIOR

Sistemes Concurrents i Paral·lels - Curs 2022-2023

Pràctica 1: Simulación de partículas (problema N-Body)

Pthreads

Informe presentado por: Alexandru Cristian Stoia Y2386362B

Pol Triquell Lombardo 48054396J

Grupo PraLab: GM2

Profesor: Manuel Fernando Cores Prado

Fecha de entrega: 14/11/2022

Grau en Enginyeria Informàtica

Universitat de Lleida

Índice de contenidos

Capítulo 1: Pthread Threads

A. *Elección de la concurrencia.*

Tras analizar todas las funciones que tiene implementadas el proyecto, nos hemos decidido por realizar una concurrencia a la hora de calcular las fuerzas que generan las partículas entre ellas.

Esta concurrencia se utilizará para calcular los valores de las fuerzas de cada partícula y, para ello, implementaremos dos funciones auxiliares cuyos nombres son CalculateForcesConcurrent y CalculateForcesThreads, en la cual al principio se llama a la función que calcula los objetos de forma secuencial, si nosotros llamamos esta misma función de forma concurrente, obtendremos un speedup.

B. *División del trabajo.*

Para obtener una división del trabajo equilibrada para cada *thread*, calcularemos la división entera entre el número de partículas y los *threads* utilizados. Una vez tenemos asignados esas partículas, si la división no es entera, tendremos partículas sin asignar, estas están calculadas con el módulo entre el número de partículas y los *threads* utilizados, asignando una partícula a cada *thread* hasta que no queden partículas por asignar.

De esta forma, cada *thread*, como máximo, tendrá una partícula extra, considerando esto una carga de trabajo dividida de forma óptima.

C. *Implementación del código.*

I. Creación threads.

Para crear los *threads*, hemos utilizado un bucle *for* y la función *pthread_create*. Con ella, se crean los *threads* y posteriormente, se hace una llamada a *CalculateForcesThreads*, que está a su vez llama a *CalculateForcesConcurrent* que hace el cálculo necesario de la fuerza de todas las partículas de los datos anteriores.

```
void CalculateForcesThreads (struct ThreadData *toDo) {

    CalculateForcesConcurrent(toDo->indexes, toDo->shrdBuff, toDo->localBuff, toDo->tree, toDo->start, toDo->end);

}
```

Como se observa, le pasamos una *struct* con los datos de los *threads*, es decir, los rangos de trabajo que debe ejecutar, el índice de las partículas en el *localBuff*, el árbol generado y los *buffers*.

II. Modificación de la función main.

a. Interfaz gráfica y no gráfica

En este momento, para poder hacer la representación de la parte no gráfica y el posterior cálculo de fuerzas de las partículas dentro del "*map*". Para ello, hemos creado el *array* de *threads* y un *array* de los atributos que tienen estos. Luego, hacemos la correcta distribución de los rangos que debe analizar cada uno. Una vez asignados, se hace el *pthread_create* para crear los *threads* correspondientes y pasarles los atributos necesarios para calcular las fuerzas de las partículas. Si la creación de estos diese error, se cancelan todos los hilos y se termina la ejecución del programa mostrando un mensaje. Una vez han acabado todos los hilos, se hace un *join* para sincronizarlos todos.

En este caso, para la implementación de la parte gráfica se ha utilizado la misma metodología.

D. *Tests de funcionamiento.*

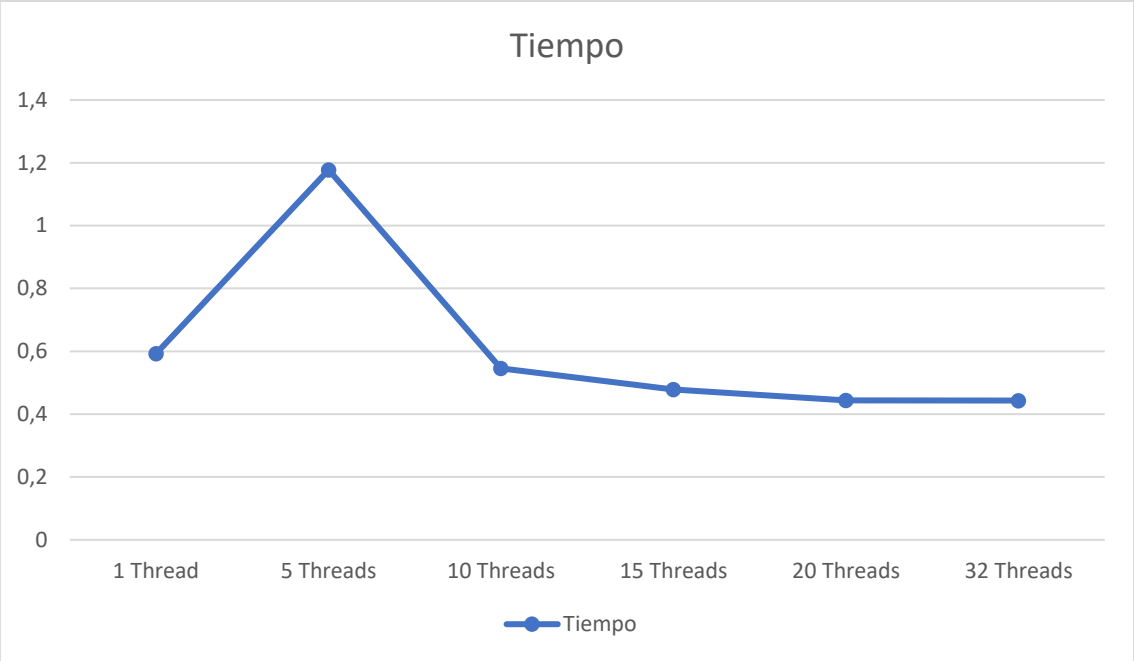
Para ver el correcto funcionamiento del código, asegurarnos que es determinista y que sigue funcionando de la misma manera que la versión original, hemos utilizados los archivos de entrada proporcionados en el campus y los hemos comparado los archivos de salida de los mismos con los nuestros, estos coinciden, por lo tanto, podemos asegurar que nuestra implementación del código no ha modificado el funcionamiento del original.

E. *Comparativa de ejecución.*

Para poder comprobar si tenemos un *speedup* real, hemos decidido hacer diferentes pruebas con un mismo *input*, utilizando un número diferente de *threads* en cada caso:

Simulación de 2000 objetos y 1000 iteraciones:

Threads	**Tiempo**
1	0m59.291s
5	1m17.744s
10	0m54.575s
15	0m47.840s
20	57.52 s
32	1 m. 4.831 s



Viendo la tabla y el gráfico, observamos claramente como el tiempo con treinta y dos *threads* es menor al resto, con un *thread* todo el trabajo se calcula con el mismo hilo, al usarlo con 5 hilos, es cuando se produce un *overhead*, provocando la ralentización del programa, donde no sale a cuenta la creación de los hilos para el procesamiento de los datos, a partir de ahí, se ve una estabilización que tiende a los 44.300 segundos.