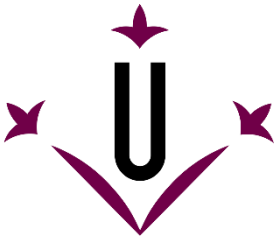
SISTEMAS CONCURRENTES Y PARALELOS

Threads en C : Pthreads



Universitat de Lleida

Universidad de Lleida
Grado en Ingeniería Informática

Marta Albets Mitjaneta

Paula Gallucci Zurita

Contenido

Introducción

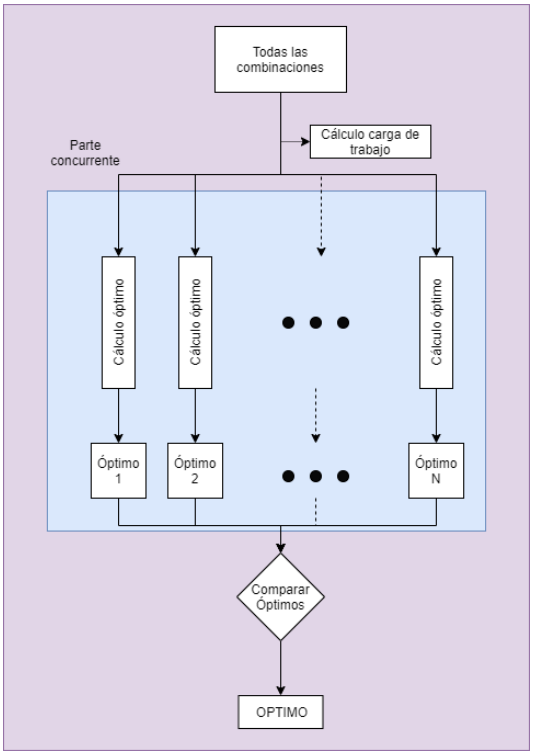
A lo largo de las siguientes páginas, se explicarán los pasos seguidos en el desarrollo de la solución concurrente al problema de los árboles y las vallas presentado en el enunciado de la práctica.

Se detallarán los algoritmos implementados adicionalmente y las modificaciones requeridas para el correcto funcionamiento del multiproceso y, finalmente, se mostrarán y explicarán los resultados de rendimiento obtenidos respecto la versión secuencial original.

El programa resultante únicamente ha sido probado en varias máquinas que hacen uso del Sistema Operativo Ubuntu 18.04 y 19.10, por lo que la funcionalidad del código en otro tipo de sistemas no está garantizada. No obstante, dado que no se han utilizado recursos adicionales, no debería encontrarse ningún fallo de ejecución en los sistemas Linux.

Diseño de la solución

El objetivo de todo desarrollo concurrente es, por definición, lograr dividir el problema original en varios subconjuntos independientes con los que poder trabajar desde distintos procesos simultáneamente, aprovechando así las características de los ordenadores *multi-core*.



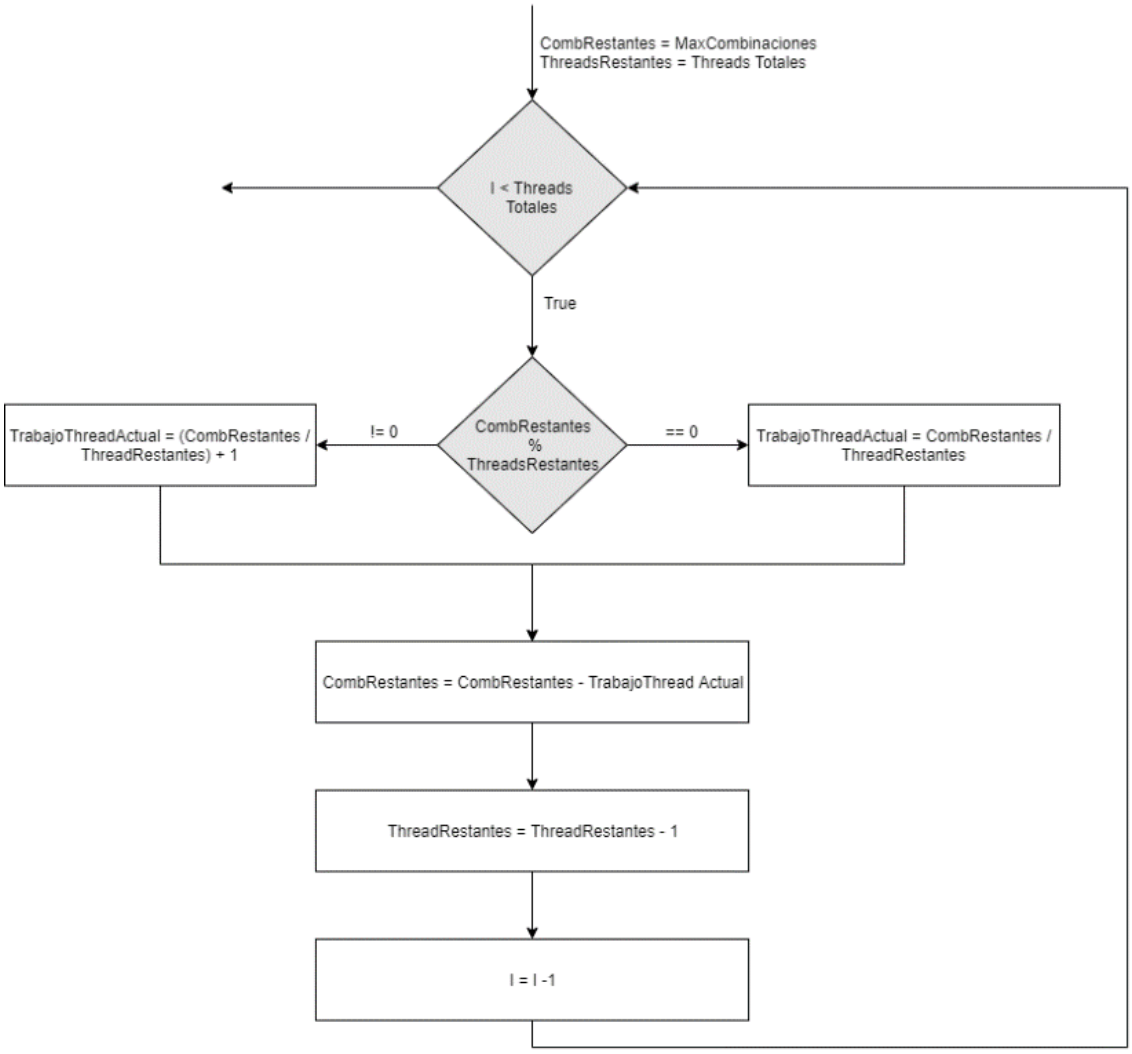
Para el problema presentado en esta práctica, hemos planteado la división del problema tras el cálculo de combinaciones totales, dividiendo equitativamente el número de combinaciones que probará cada uno de los procesos (o *threads*) que conformarán el programa. Cada uno de estos subconjuntos de combinaciones devolverá un óptimo, por lo que finalmente se deberán comparar dichos valores resultantes con la finalidad de obtener el óptimo global.

En el dibujo adjunto, se puede ver el funcionamiento básico de la estructura; funcionamiento sobre el que se profundizará en las siguientes páginas.

Cálculo de carga de trabajo

El correcto cómputo a la hora de dividir el trabajo de cada subproceso es, sin lugar a dudas, uno de los puntos clave en toda ejecución concurrente. Repartir el trabajo de forma **no** equitativa podría suponer no llegar a una solución con un rendimiento óptimo o, incluso, no notar mejoría o tener un peor rendimiento respecto una ejecución secuencial del mismo problema.

Para lograr un reparto equitativo del número total de combinaciones, se ha implementado un simple algoritmo que calcula la cantidad correcta de trabajo al thread actual:



De esta manera, podemos conseguir repartir el trabajo de forma equitativa donde, como máximo, habrá una diferencia de una combinación entre el thread con mayor cantidad de trabajo y el de menor cantidad de trabajo.

Obtención del óptimo parcial

Para calcular el óptimo de cada subconjunto de combinaciones perteneciente a cada thread, se ha hecho uso de la función ya implementada en la solución secuencial *CalcularCombinacionOptima*.

Obtención del óptimo global

Únicamente tras finalizar el último de los threads en ejecución, se procederá a la comparación entre ellos con el objetivo de obtener el óptimo entre los óptimos parciales de cada uno.

Para esto, se hará una comparación entre todos sus costes, asignando como Óptimo aquel de menor coste.

Implementación

Gracias a disponer previamente de la solución secuencial, la implementación de una versión concurrente ha sido relativamente sencilla.

Los cambios más cruciales realizados respecto la solución secuencial son:

- Implementación de una nueva estructura: *ParamsThread* que contendrá los parámetros de cada thread:
  - Primera combinación
  - Última combinación
  - Óptimo parcial

- Implementación del algoritmo de asignación de carga de trabajo: se ha implementado en la función *CalculoCargaDeTrabajo*

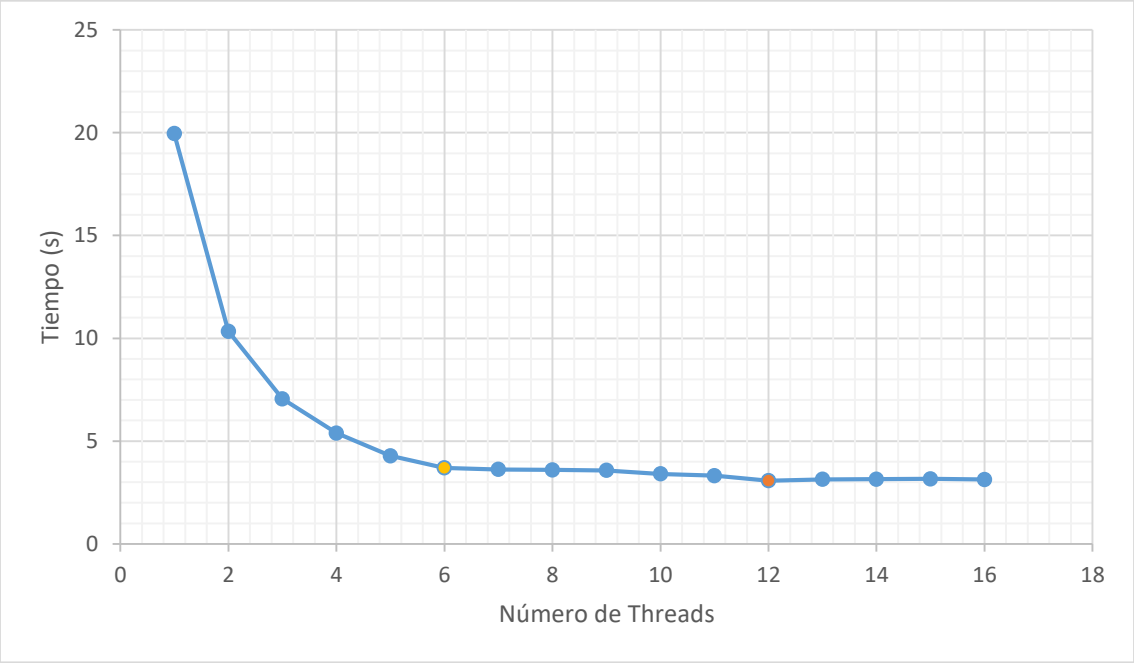
Resultados

Las pruebas de eficiencia de la solución se han probado en una máquina de las siguientes características:

- Sistema Operativo: Windows 10 desde Bash Ubuntu
- Memoria: 16Gb DDR4
- Procesador: Intel i7-8700 3,20GHz, 6 núcleos. 12 subprocesos.

Con el objetivo de reducir el tiempo de ejecución al máximo, se han eliminado las impresiones por pantalla durante la ejecución.

Se ha probado la ejecución de “EjemploGordo” con varios hilos y, a continuación, se muestra la gráfica de rendimiento obtenida de todas las ejecuciones y, seguidamente, los resultados individuales.



Observaciones

Se puede observar un aumento de rendimiento mayor a medida que se aumentan los threads hasta llegar a 6, que corresponde con el número de núcleos físicos del procesador de la máquina.

Tras esta cantidad, el tiempo de ejecución sigue disminuyendo, a menor velocidad, hasta llegar a 12, que es el número de núcleos virtuales de la máquina.

Una vez alcanzados los doce procesos simultáneos, se puede observar como un aumento en la cantidad de threads simultáneos no implica un aumento de rendimiento en la velocidad de ejecución.

Además, hay que tener en consideración que el programa de la práctica debe compartir procesador con otros procesos, tanto del sistema como del usuario.

1 Thread (Secuencial)

Tiempo de ejecución aproximado: **19,958s**

Combinaciones por segundo: **1.681.252 combinaciones**

```
latra@pc-1563296879:/mnt/d/Universidad/SistemasConcurrentesx/SistemasConcurrentes$ time ./practica EjemploGordo.dat 1
El coste optimo es 22.000000

real    0m19.958s
```

6 Threads (Número de núcleos físicos)

Tiempo de ejecución aproximado: **3,703s**

Combinaciones por segundo: **9.061.418 combinaciones**

```
latra@pc-1563296879:/mnt/d/Universidad/SistemasConcurrentesx/SistemasConcurrentes$ time ./practica EjemploGordo.dat 6
El coste optimo es 22.000000

real    0m3.703s
```

12 Threads (Número de núcleos virtuales)

Tiempo de ejecución aproximado: **3,075s**

Combinaciones por segundo: **10.799.624 combinaciones**

```
latra@pc-1563296879:/mnt/d/Universidad/SistemasConcurrentesx/SistemasConcurrentes$ time ./practica EjemploGordo.dat 12

El coste optimo es 22.000000

real    0m3.075s
```

16 Threads

Tiempo de ejecución aproximado: **3,174s**

Combinaciones por segundo: **10.740.855 combinaciones**

```
latra@pc-1563296879:/mnt/d/Universidad/SistemasConcurrentesx/SistemasConcurrentes$ time ./practica EjemploGordo.dat 16
El coste optimo es 22.000000

real    0m3.124s
```