UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Máster Universitario en Computación en la Nube y de Altas Prestaciones

Tecnología de la Programación Paralela (TPP).

Ejercicios OpenMP Sesión 3

Ejercicio 1: Paralelización de bucles en OpenMP con tareas

En este ejercicio se va a realizar la paralelización del bucle del código del fichero `bucle.c` utilizando un **diseño basado en tareas**, es decir, utilizando la directiva `task` de OpenMP.

El programa y su paralelización han de ser estudiadas. Aunque no es necesario presentar una memoria para estos ejercicios, síes necesario ser capaz de estudiar la paralelización. Se recomienda trabajar con tamaños, por ejemplo, de unos 40000 vectores de un tamaño máximo de 40000. El estudio del algoritmo paralelo debe consistir en mostrar la evolución de los tiempos cuando el número de cores es de 1, 2, 4, 8, 16 y 24. Además del tiempo es necesario sacar el incremento de velocidad o *speedup* y la eficiencia para cada caso. Realizado el mismo estudio, con números, que el efectuado en el mismo ejercicio de la primera sesión, comparar las diferencias entre ambos enfoques de paralelización.

Ejercicio 2: El problema de la mochila con tareas

Para tratar este caso de estudio vamos a comenzar por un algoritmo secuencial eficiente en lugar de utilizar el algoritmo de fuerza bruta visto anteriormente. Sea $F(i, c)$ el máximo valor de una mochila de capacidad c utilizando i objetos, entonces, el problema se puede definir como un problema de *programación dinámica* del siguiente modo:

$$F(i,c) = \begin{cases} -\infty & c < 0, \\ 0 & c \geq 0, \quad i = 0, \\ \text{máx}\{F(i-1,c), (F(i-1,c-w_i)+p_i)\} & c \geq 0, \quad 1 \leq i \leq n. \end{cases}$$

Esta ecuación recursiva conduce a una mochila con el máximo valor. Cuando la capacidad actual es c, la decisión de incluir el objeto i puede llevar a dos situaciones: (i) el objeto no se incluye, la capacidad de la mochila sigue siendo c y el valor permanece inalterable; (ii) el objeto es incluido, la capacidad pasa a ser $c - w_i$ y el valor aumenta en p_i.

El fichero `knapsack.c` contiene una implementación secuencial del problema. Se puede probar compilando con

```
gcc -o knapsack knapsack.c ctimer.c
```

o añadiendo la opción `-DCHECK` en la línea de compilación si se quiere ver la relación de pesos y valores de las piedras y la solución.

Este tipo de algoritmo se le conoce también como de *backtracking* porque las llamadas recursivas van construyendo un árbol en profundidad hasta llegar a las hojas (caso base de la recursión) para retornar después hacia arriba, hasta la raíz, donde se obtiene la solución. La Figura 1 muestra el árbol de llamadas para un ejemplo con 3 piedras de pesos $\{3, 7, 3\}$ y una capacidad de $c = 13$ ($F(3, 13)$). En este ejemplo, los valores de las piedras no son importantes ya que caben todas en la mochila, por lo tanto, la solución está clara. El ejemplo sirve para observar la ejecución del algoritmo cuando se desarrollan todas las posibilidades y, por tanto, el árbol generado es binario y completo.

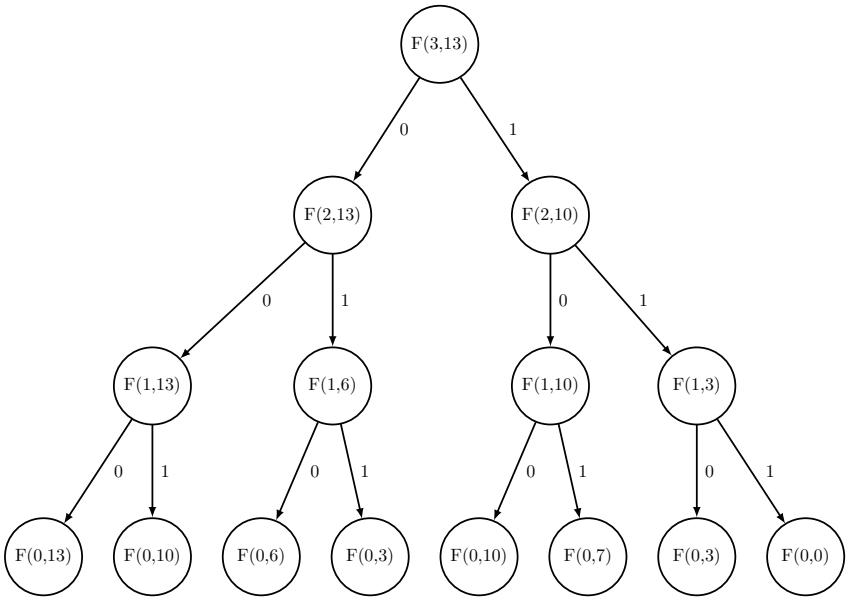

Figura 1: DAG correspondiente al problema $F(3, 13)$.

En la Figura 2 se muestra el mismo ejemplo con una mochila de capacidad $c = 9$, por lo tanto, ya no caben todas las piedras y hay que elegir. Este hecho implica que el árbol generado no es completo ya que se hace "poda" cuando la piedra no se puede introducir en la mochila. Los arcos y los nodos coloreados en rojo (capacidad negativa) se muestran por claridad ya que, en realidad, no se generan.

Tarea a realizar: Implementar en paralelo un "diseño basado en tareas" de la rutina `kanpsack` proporcionada en el fichero `knapsack.c`. Esta rutina no devuelve el vector solución, solo el valor máximo obtenido. La parte compleja del análisis está en saber cuándo cortar la generación de tareas concurrentes. Adicionalmente, se puede realizar una implementación de la versión que devuelve el vector solución (rutina `knapsackv`).

Ejercicio 3: Paralelización del Sudoku basada en tareas

La tarea a realizar consiste en volver al ejemplo del Sudoku y realizar una paralelización del mismo **basada en tareas**.

Para realizar esta tarea es necesario partir del código secuencial inicial que resuelve el Sudoku. A este código hay que añadir aquellas directivas de OpenMP relacionadas con tareas (`task`).

Una vez realizada la implementación del algoritmo, hay que realizar una comparación con la versión basada en hilos implementada anteriormente. En este caso, no existe `profundidad` ni tampoco un bucle para el que variar la política de planificación, luego, en este caso, solo tenemos un tiempo de ejecución paralelo. Realizar una comparativa del tiempo de ejecución con 1, 2, 4, 8, 16 y 24 hilos con el mejor algoritmo paralelo obtenido mediante la estrategia de paralelización "basada en hilos".

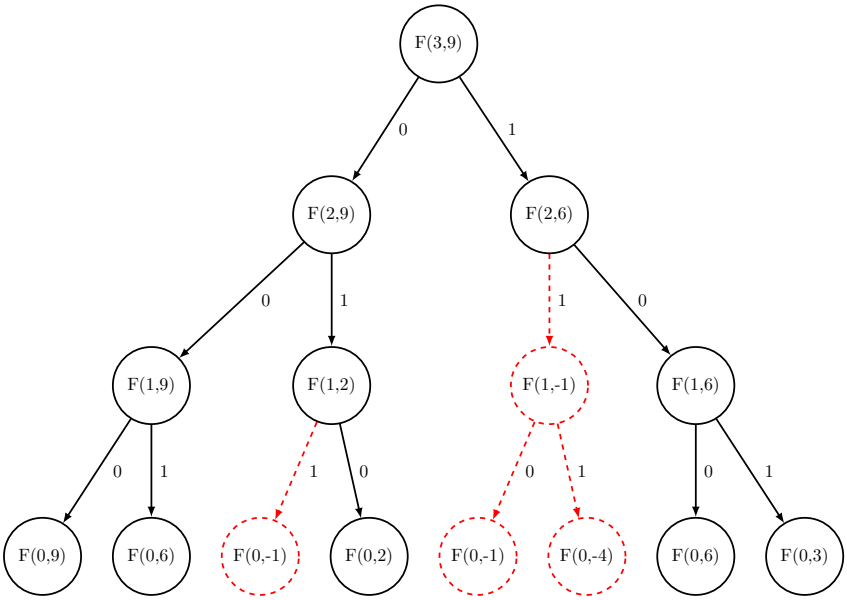

Figura 2: DAG correspondiente al problema $F(3, 9)$.