EXAMEN DE CONCURRENCIA Y PARALELISMO

Bloque II: Paralelismo

APELLIDOS: _____ **NOMBRE:** _____

Convocatoria ordinaria

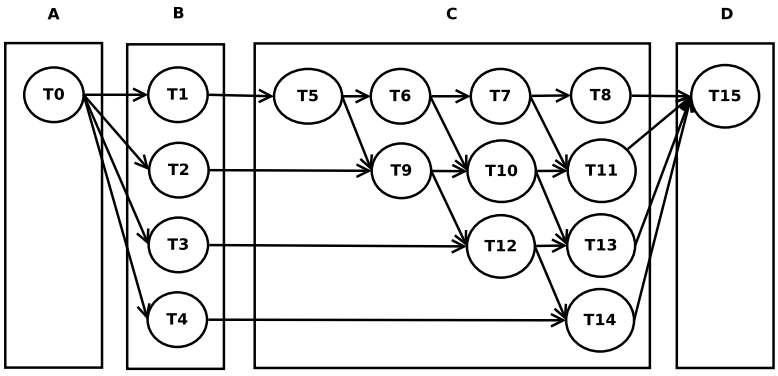
21 de mayo de 2019

AVISO: *No mezcleis en el mismo folio preguntas del bloque de concurrencia y del bloque de paralelismo. Las respuestas a cada bloque se entregan por separado.*

[2,5p] 1. **Conceptos de paralelismo.** Considera un programa con cuatro regiones de código (A, B, C y D) ejecutadas una tras otra en su versión secuencial inicial. Las regiones A y D no se pueden paralelizar, necesitando 200 y 300 operaciones, respectivamente. Por el contrario, las regiones B y C sí pueden ser paralelizadas, realizando 16.000 y 25.000 operaciones, respectivamente.

(a) Calcula, según la ley de Amdhal, cuál es la máxima aceleración que puede obtener este programa. [0,5p].

(b) Tras aplicar la descomposición de tareas a las regiones B y C, nos damos cuentas de que la región B puede ser ejecutada completamente en paralelo mientras que la región C sí presenta dependencias entre algunas de sus tareas. La siguiente figura muestra el grafo de dependencias. Sabiendo que las tareas de la región B necesitan 4.000 operaciones cada una, mientras que las de la región C necesitan 2.500 operaciones cada una, ¿cuál es el grado medio de concurrencia? [0,75p]



(c) Si las tareas son asignadas a cuatro procesos como sigue: P0={T0, T1, T5, T6, T7, T8, T15}, P1={T2, T9, T10, T11}, P2={T3, T12, T13} y P3={T4, T14} ¿cuál sería el speedup que podría obtenerse? ¿y la eficiencia paralela? [0,5p]

(d) De cara a mejorar el speedup obtenido con cuatro procesos, se propone dividir de nuevo CADA UNA de las tareas de la región B en cuatro subtareas completamente independientes. Por ejemplo, T1 sería reemplazada por T1.1, T1.2, T1.3 y T1.4, cada una con 1.000 operaciones y con T5 dependiendo de todas ellas (de forma análoga para T2, T3 y T4, con T9, T12 y T14 dependiendo de ellas, respectivamente). ¿Se mejora el grado medio de concurrencia con esta aproximación? Proporciona una asignación de tareas que obtenga el mejor speedup posible para cuatro procesos e indica el valor de ese speedup. [0,75p]

SOLUCIÓN

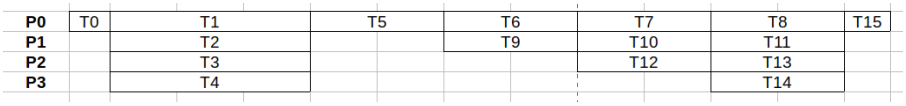
(a) El problema tiene una parte intrínsecamente secuencial con 500 operaciones y otra perfectamente paralelizable con 41.000 operaciones. Aplicando la Ley de Amdahl:

$$sp(p \to \infty) = \frac{T_{sec}}{T_{par}(p \to \infty)} = \lim_{p \to \infty} \frac{500 + 41.000}{500 + \frac{41.000}{p}} = \frac{41.500}{500} = 83$$

(b) El camino crítico comprende a T0 (200 operaciones), una de las subtareas de la región B (4.000 operaciones), cuatro de las subtareas de la región C (10.000 operaciones) y T15 (300 operaciones). Por tanto el camino crítico necesita 14.500 operaciones. El grado medio de concurrencia es el número total de operaciones entre la longitud del camino crítico:

$$\frac{41.500}{14.500} \simeq 2.86$$

(c) El siguiente gráfico muestra la distribución de tareas entre los procesos.



El tiempo paralelo es el tiempo del proceso con más operaciones (P0): 14.500 operaciones. Por tanto:

$$Sp(4) = \frac{41.500}{14.500} \simeq 2.86$$

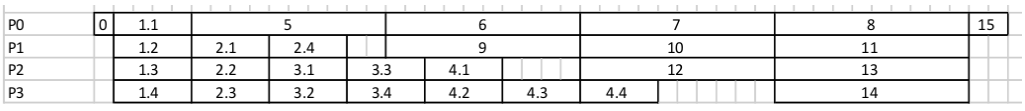
$$Eff(4)\frac{2.86}{4} \simeq 0.72$$

(d) Sí se mejora el grado medio de concurrencia porque ahora no toda la T1 está en el camino crítico sino solo un cuarto de ella. Por tanto este camino crítico tiene una longitud de 11.500 operaciones en lugar de 14.500. Esto indica un grado medio de concurrencia de:

$$\frac{41.500}{11.500} \simeq 3.61$$

A continuación se muestra una asignación que logra un speedup máximo (igual al grado medio de concurrencia):

- P0 = T0, T1.1, T5, T6, T7, T8, T15
- P1 = T1.2, T2.1, T2.4, T9, T10, T11
- P2 = T1.3, T2.2, T3.1, T3.3, T4.1, T12, T13
- P3 = T1.4, T2.3, T3.2, T3.4, T4.2, T4.3, T4.4, T14



[2,5p] 2. **Diseño de algoritmos paralelos.** Queremos paralelizar el siguiente código que trabaja con matrices 2D siguiendo el llamado algoritmo *Rojo-Negro*.

```
void main(int argc, char *argv[]) {
    double tmp;
    double u[N][N];
    LeeMatriz(u);
    // Bucle rojo: atravesando todas las celdas rojas
    for(int i=2; i<N-1; i+=2)
        for(int j=1; j<N-1; j++){
            tmp = u[i+1][j]+u[[i-1][j]+u[i][j+1]+u[i][j-1]-4*u[i][j];
            u[i][j] = tmp/4;
        }

    // Bucle negro: atravesando todas las celdas negras
    for(int i=1; i<N-1; i+=2)
        for(int j=1; j<N-1; j++){
            tmp = u[i+1][j]+u[[i-1][j]+u[i][j+1]+u[i][j-1]-4*u[i][j];
            u[i][j] = tmp/4;
        }

    EscribeMatriz(u);
}
```

(a) Dibuja una matriz de dimensiones 9x9 que represente a u en el caso de que N es igual a 9. Indica para cada celda si es roja (R), negra (N) o blanca (B, aquellas celdas que no se calculan). [0,5p]

(b) ¿Existen dependencias dentro del bucle rojo? ¿Y dentro del negro? ¿Y entre los dos bucles? Razona tus respuestas. [0,5p]

(c) ¿Qué tipo de descomposición de tareas debemos hacer? ¿Qué tipo de asignación de tareas? [0,5p]

(d) Teniendo en cuenta tus respuestas a los apartados anteriores, para una ejecución con tres procesos y N igual a 9 indica a qué elementos debe acceder cada uno de los procesos. Puedes usar una representación gráfica. [0,5p]

(e) Para una ejecución como la del apartado anterior, indica qué elementos deben comunicarse entre que se termina el bucle rojo y comienza el negro. Para cada mensaje debes indicar las filas/columnas enviadas, el tamaño del mensaje, el proceso emisor, y el receptor. [0,5p]

SOLUCIÓN

B	B	B	B	B	B	B	B	B
B	N	N	N	N	N	N	N	B
B	R	R	R	R	R	R	R	B
B	N	N	N	N	N	N	N	B
B	R	R	R	R	R	R	R	B
B	N	N	N	N	N	N	N	B
B	R	R	R	R	R	R	R	B
B	N	N	N	N	N	N	N	B
B	B	B	B	B	B	B	B	B

(a)

(b) Las dependencias dentro de un mismo bucle son por columnas, ya que para calcular el elemento (i, j) se necesita el valor del elemento (i, j-1) que ha sido calculado antes. Las dependencias de filas a causa del elemento (i-1, j) no se dan dentro del mismo bucle (ya que cada uno se salta una fila) sino entre los dos bucles.

(c) Se realizan los mismos cálculos sobre todos los elementos negros y rojos, por tanto una descomposición de dominio sobre la matriz u es adecuada para este algoritmo. Se podría llevar a cabo tanto por filas como por columnas, elegimos la primera porque suele ser más fácil de implementar posterirmente con lenguajes basados en C (con los elementos almecenados en memoria de forma consecutiva por filas). Cada fila sería una tarea.

A la hora de asignar, como existen dependencias de filas entre los bucles, lo más lógico es hacer una asignación bloque donde filas consecutivas pertenezcan a un mismo proceso. Esto puede desbalancear un poco ya que la primera y última fila (asignadas al primer y último proceso, respectivamente) no realizan cálculo. Pero este desbalanceo no debería ser importante en tamaños grandes de matrices. Para poder hacer todos los cálculos los procesos necesitan las filas inmediatamentes superiores e inferiores a las que tienen que calcular.

(d) Cada uno de los procesos calcula tres filas de forma bloque. El primer proceso tendría que acceder a las cuatro primeras filas (la cuarta es necesaria para poder calcular la tercera). De forma análoga el último proceso debe acceder a las cuatro últimas filas. El proceso del medio debe acceder a las filas 2, 3, 4, 5 y 6, para poder calcular las filas 3, 4 y 5.

(e) Como el bucle rojo es anterior al bucle negro solo hay que fijarse en las dependencias rojo-negro. En este caso hay dos que involucran mensajes. Por un lado P1 necesita tener actualizada la fila 2 (calculada por P0). Eso se traduce en que P0 manda a P1 los siete elementos centrales de la fila 2. También P1 necesita tener actualizada la fila 6 para poder calcular la fila 5. Eso se traduce en que P2 manda a P1 los siete elementos centrales de la fila