



2º curso / 2º cuatr.

Grados en
Ing. Informática

Arquitectura de Computadores. Ejercicios y cuestiones

Tema 2. Programación paralela

Material elaborado por: Mancia Anguita, Julio Ortega

1 Ejercicios

Ejercicio 1. Un programa tarda 40 s en ejecutarse en un multiprocesador. Durante un 20% de ese tiempo se ha ejecutado en cuatro procesadores; durante un 60%, en tres; y durante el 20% restante, en un procesador (consideramos que se ha distribuido la carga de trabajo por igual entre los procesadores que colaboran en la ejecución en cada momento, despreciamos sobrecarga). ¿Cuánto tiempo tardaría en ejecutarse el programa en un único procesador? ¿Cuál es la ganancia en velocidad obtenida con respecto al tiempo de ejecución secuencial? ¿Y la eficiencia?

Ejercicio 2. Un programa tarda 20 s en ejecutarse en un procesador P1, y requiere 30 s en otro procesador P2. Si se dispone de los dos procesadores para la ejecución del programa (despreciamos sobrecarga):

- ¿Qué tiempo tarda en ejecutarse el programa si la carga de trabajo se distribuye por igual entre los procesadores P1 y P2?
- ¿Qué distribución de carga entre los dos procesadores P1 y P2 permite el menor tiempo de ejecución utilizando los dos procesadores en paralelo? ¿Cuál es este tiempo?

Ejercicio 3. ¿Cuál es fracción de código paralelo de un programa secuencial que, ejecutado en paralelo en 8 procesadores, tarda un tiempo de 100 ns, durante 50ns utiliza un único procesador y durante otros 50 ns utiliza 8 procesadores (distribuyendo la carga de trabajo por igual entre los procesadores)?

Ejercicio 4. Un 25% de un programa no se puede parallelizar, el resto se puede distribuir por igual entre cualquier número de procesadores. ¿Cuál es el máximo valor de ganancia de velocidad que se podría conseguir al parallelizarlo en p procesadores, y con infinitos? ¿A partir de cuál número de procesadores se podrían conseguir ganancias mayores o iguales que 2?

Ejercicio 5. En la Figura 1, se presenta el grafo de dependencia entre tareas para una aplicación. La figura muestra la fracción del tiempo de ejecución secuencial que la aplicación tarda en ejecutar las tareas del grafo. Suponiendo un tiempo de ejecución secuencial de 60 s, que las tareas no se pueden dividir en tareas de menor granularidad y que el tiempo de comunicación es despreciable, obtener el tiempo de ejecución en paralelo y la ganancia en velocidad en un computador con:

- 4 procesadores.
- 2 procesadores.

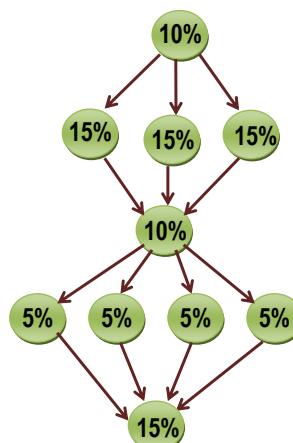


Figura 1. Grafo de tareas del ejercicio 5.

Ejercicio 6. Un programa se ha conseguido dividir en 10 tareas. El orden de precedencia entre las tareas se muestra con el grafo dirigido de la Figura 2. La ejecución de estas tareas en un procesador supone un tiempo de 2 sg. El 10% de ese tiempo es debido a la ejecución de la tarea 1; el 15% a la ejecución de la tarea 2; otro 15% a la ejecución de 3; cada tarea 4, 5, 6 o 7 supone el 9%; un 8% supone la tarea 8; la tarea 9 un 10%; por último, la tarea 10 supone un 6%. Se dispone de una arquitectura con 8 procesadores para ejecutar la aplicación. Consideramos que el tiempo de comunicación se puede despreciar. **(a)** ¿Qué tiempo tarda en ejecutarse el programa en paralelo? **(b)** ¿Qué ganancia en velocidad se obtiene con respecto a su ejecución secuencial?.

Ejercicio 7. Se quiere parallelizar el siguiente trozo de código:

```
{Cálculos antes del bucle}
for( i=0; i<w; i++) {
    Código para i
}
{cálculos después del bucle}
```

Los cálculos antes y después del bucle suponen un tiempo de t_1 y t_2 , respectivamente. Una iteración del ciclo supone un tiempo t_i . En la ejecución paralela, la inicialización de p procesos supone un tiempo k_1p (k_1 constante), los procesos se comunican y se sincronizan, lo que supone un tiempo k_2p (k_2 constante); k_1p+k_2p constituyen la sobrecarga.

- (a)** Obtener una expresión para el tiempo de ejecución paralela del trozo de código en p procesadores (T_p).
- (b)** Obtener una expresión para la ganancia en velocidad de la ejecución paralela con respecto a una ejecución secuencial (S_p).
- (c)** ¿Tiene el tiempo T_p con respecto a p una característica lineal o puede presentar algún mínimo? ¿Por qué? En caso de presentar un mínimo, ¿para qué número de procesadores p se alcanza?

Ejercicio 8. Supongamos que se va a ejecutar en paralelo la suma de n números en una arquitectura con p procesadores o cores (p y n potencias de dos) utilizando un grafo de dependencias en forma de árbol (divide y vencerás) binario para las tareas.

- (a)** Dibujar el grafo de dependencias entre tareas para $n = 16$ y p igual a 8. Hacer una asignación de tareas a procesos.
- (b)** Obtener el tiempo de cálculo paralelo para cualquier n y p con $n > p$ suponiendo que se tarda una unidad de tiempo en realizar una suma.
- (c)** Obtener el tiempo comunicación del algoritmo suponiendo (1) que las comunicaciones en un nivel del árbol se pueden realizar en paralelo en un número de unidades de tiempo igual al número de datos que recibe o envía un proceso en cada nivel del grafo de tareas (tenga en cuenta la asignación de tareas a procesos que ha considerado en el apartado (a)) y (2) que los procesadores que realizan las tareas de las hojas del árbol tienen acceso sin coste de comunicación a los datos que utilizan dichas tareas.
- (d)** Suponiendo que el tiempo de sobrecarga coincide con el tiempo de comunicación calculado en (c), obtener la ganancia en prestaciones.
- (e)** Obtener el número de procesadores para el que se obtiene la máxima ganancia con n números.

Ejercicio 9. Se va a parallelizar un decodificador JPEG en un multiprocesador. Se ha extraído para la aplicación el siguiente grafo de tareas que presenta una estructura segmentada (o de flujo de datos):

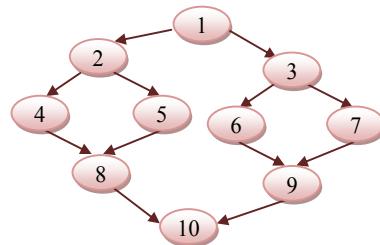
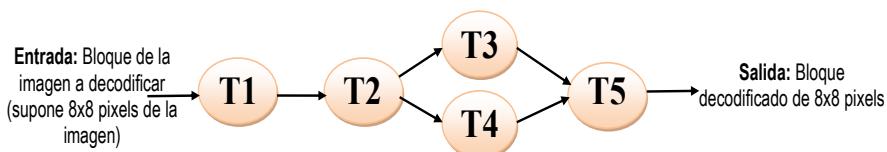


Figura 2. Grafo de dependencia entre tareas del ejercicio 6.



La tareas 1, 2 y 5 se ejecutan en un tiempo igual a t , mientras que las tareas 3 y 4 suponen 1,5t

El decodificador JPEG aplica el grafo de tareas de la figura a bloques de la imagen, cada uno de 8x8 píxeles. Si se procesa una imagen que se puede dividir en n bloques de 8x8 píxeles, a cada uno de esos n bloques se aplica el grafo de tareas de la figura. Obtenga la mayor ganancia en prestaciones que se puede conseguir paralelizando el decodificador JPEG en (suponga despreciable el tiempo de comunicación/sincronización): **(a)** 5 procesadores, y **(b)** 4 procesadores. En cualquier de los dos casos, la ganancia se tiene que calcular suponiendo que se procesa una imagen con un total de n bloques de 8x8 píxeles.

Ejercicio 10. Se quiere implementar un programa paralelo para un multicomputador que calcule la siguiente expresión para cualquier x (es el polinomio de interpolación de Lagrange): $P(x) = \sum_{i=0}^n (b_i \cdot L_i(x))$, donde

$$L_i(x) = \frac{(x - a_0) \cdot \dots \cdot (x - a_{i-1}) \cdot (x - a_{i+1}) \cdot \dots \cdot (x - a_n)}{k_i} = \frac{\prod_{\substack{j=0 \\ j \neq i}}^n (x - a_j)}{k_i} \quad i = 0, 1, \dots, n$$

$$k_i = (a_i - a_0) \cdot \dots \cdot (a_i - a_{i-1}) \cdot (a_i - a_{i+1}) \cdot \dots \cdot (a_i - a_n) = \prod_{\substack{j=0 \\ j \neq i}}^n (a_i - a_j) \quad i = 0, 1, \dots, n$$

Inicialmente k_i , a_i y b_i se encuentra en el nodo i y x en todos los nodos. Sólo se van a usar funciones de comunicación colectivas. Indique cuál es el número mínimo de funciones colectivas que se pueden usar, cuáles serían, en qué orden se utilizarían y para qué se usan en cada caso.

Ejercicio 11. (a) Escriba un programa secuencial con notación algorítmica (podría escribirlo en C) que determine si un número de entrada, x , es primo o no. El programa imprimirá si es o no primo. Tendrá almacenados en un vector, NP , los M números primos entre 1 y el máximo valor que puede tener un número de entrada al programa.

(b) Escriba una versión paralela del programa anterior para un multicomputador usando un estilo de programación paralela de paso de mensajes. El proceso 0 tiene inicialmente el número x y el vector NP en su memoria e imprimirá en pantalla el resultado. Considere que la herramienta de programación ofrece funciones `send()`/`receive()` para implementar una comunicación uno-a-uno asíncrona, es decir, con función `send(buffer, count, datatype, idproc, group)` no bloqueante y `receive(buffer, count, datatype, idproc, group)` bloqueante. En las funciones `send()`/`receive()` se especifica:

- `group`: identificador del grupo de procesos que intervienen en la comunicación.
- `idproc`: identificador del proceso al que se envía o del que se recibe.
- `buffer`: dirección a partir de la cual se almacenan los datos que se envían o los datos que se reciben.
- `datatype`: tipo de los datos a enviar o recibir (entero de 32 bits, entero de 64 bits, flotante de 32 bits, flotante de 64 bits, ...).
- `count`: número de datos a transferir de tipo `datatype`.

Ejercicio 12. Escribir una versión paralela del programa secuencial del ejercicio 11 para un multicomputador usando un estilo de programación paralela de paso de mensajes y suponiendo que la herramienta de programación ofrece las funciones colectivas de difusión y reducción (escribir primero la versión secuencial).

Sólo el proceso 0 imprimirá en pantalla. En la función de difusión, `broadcast(buffer, count, datatype, idproc, group)`, se especifica:

- `group`: identificador del grupo de procesos que intervienen en la comunicación, todos los procesos del grupo reciben.
- `idproc`: identificador del proceso que envía.
- `buffer`: dirección de comienzo en memoria de los datos que difunde `idproc` y que almacenará, en todos los procesos del grupo, los datos difundidos.
- `datatype`: tipo de los datos a enviar/recibir (entero de 32 bits, entero de 64 bits, flotante de 32 bits, flotante de 64 bits ...).
- `count`: número de datos a transferir de tipo `datatype`.

En la función de reducción, `reduction(sendbuf, recvbuf, count, datatype, oper, idproc, group)`, se especifica:

- `group`: identificador del grupo de procesos que intervienen en la comunicación, todos los procesos del grupo envían.
- `idproc`: identificador del proceso que recibe.
- `recvbuf`: dirección en memoria a partir de la cual se almacena el escalar resultado de la reducción de todos los componentes de todos los vectores `sendbuf`.
- `sendbuf`: dirección en memoria a partir de la cual almacenan todos los procesos del grupo los datos de tipo `datatype` a reducir (uno o varios).
- `datatype`: tipo de los datos a enviar y recibir (entero de 32 bits, entero de 64 bits, flotante de 32 bits, flotante de 64 bits, ...).
- `oper`: tipo de operación de reducción. Puede tomar los valores OR, AND, ADD, MUL, MIN, MAX
- `count`: número de datos de tipo `datatype`, del buffer `sendbuffer` de cada proceso, que se van a reducir.

Ejercicio 13. Escriba una versión paralela del programa secuencial del Ejercicio 11 para un multiprocesador usando el estilo de programación paralela de variables compartidas; en particular, use OpenMP (escriba primero la versión secuencial).

2 Cuestiones

Cuestión 1. Indique las diferencias entre OpenMP y MPI.

Cuestión 2. Ventajas e inconvenientes de una asignación estática de tareas a procesos/threads frente a una asignación dinámica.

Cuestión 3. ¿Qué se entiende por escalabilidad lineal y por escalabilidad superlineal? Indique las causas por las que se puede obtener una escalabilidad superlineal.

Cuestión 4. Enuncie la ley de Amdahl en el contexto de procesamiento paralelo.

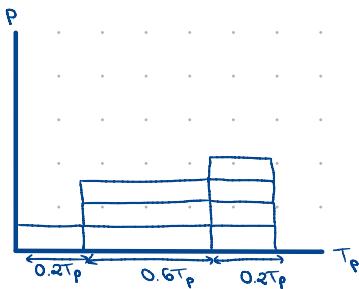
Cuestión 5. Deduzca la expresión matemática que se suele utilizar para caracterizar la ley de Gustafson. Defina claramente y sin ambigüedad el punto de partida que va a utilizar para deducir esta expresión y cada una de las etiquetas que utilice. ¿Qué nos quiere decir Gustafson con esta ley?

Cuestión 6. Deduzca la expresión que caracteriza a la ley de Amdahl. Defina claramente el punto de partida y todas las etiquetas que utilice.

Ejercicio 1. Un programa tarda 40 s en ejecutarse en un multiprocesador. Durante un 20% de ese tiempo se ha ejecutado en cuatro procesadores; durante un 60%, en tres; y durante el 20% restante, en un procesador (consideramos que se ha distribuido la carga de trabajo por igual entre los procesadores que colaboran en la ejecución en cada momento, despreciamos sobrecarga). ¿Cuánto tiempo tardaría en ejecutarse el programa en un único procesador? ¿Cuál es la ganancia en velocidad obtenida con respecto al tiempo de ejecución secuencial? ¿Y la eficiencia?

$$T_p = 40 \text{ s} \quad T_p = T_c + T_o \xrightarrow{\text{D}}$$

¿ T_s ? $\leftarrow S(4) \rightleftharpoons E(4)$?



$$T_s = 0.2T_p + 3 \cdot 0.6T_p + 4 \cdot 0.2T_p = 2.8 \cdot 40 \text{ ns} = 112 \text{ ns}$$

$$\frac{\text{Prest}(p)}{\text{Prest}(1)} = \frac{S(p)}{T_p(1)} = \frac{T_s}{T_p(1)} = \frac{2.8T_p}{T_p} = 2.8 < 4 \xrightarrow{\text{Maxima ganancia ideal}}$$

$$E(4) = \frac{\text{Prest}(p)}{p \cdot \text{Prest}(1)} = \frac{S(p)}{p} = \frac{2.8}{4} = 0.7 < 1$$

Máxima eficiencia

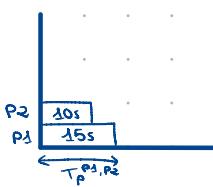
Ejercicio 2. Un programa tarda 20 s en ejecutarse en un procesador P1, y requiere 30 s en otro procesador P2. Si se dispone de los dos procesadores para la ejecución del programa (despreciamos sobrecarga):

- ¿Qué tiempo tarda en ejecutarse el programa si la carga de trabajo se distribuye por igual entre los procesadores P1 y P2?
- ¿Qué distribución de carga entre los dos procesadores P1 y P2 permite el menor tiempo de ejecución utilizando los dos procesadores en paralelo? ¿Cuál es este tiempo?

a)

$$\begin{aligned} T_{P1}^{P1} &= 20 \text{ s} \\ T_{P2}^{P2} &= 30 \text{ s} \end{aligned}$$

$$\begin{aligned} T_p^{P1, P2} &= \frac{20 \text{ s}}{2} = 10 \text{ s} \\ T_p^{P1, P2} &= \frac{30 \text{ s}}{2} = 15 \text{ s} \end{aligned} \quad \left\{ \begin{array}{l} T_p^{P1, P2} = \max(T_p^{P1}, T_p^{P2}) = 15 \text{ s} \end{array} \right.$$



b)

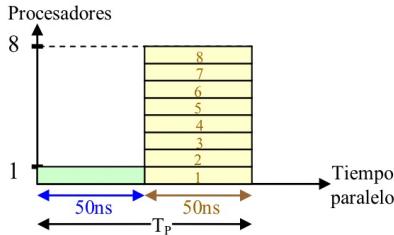
P1	P2
x	$1-x$

$$T_p^{P1}(x) = T_p^{P2}(1-x) \Rightarrow 20s \cdot x = 30s \cdot (1-x) \Rightarrow x = \frac{3}{5}$$

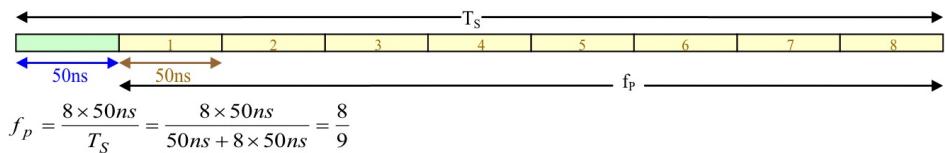
Ejercicio 3. ¿Cuál es fracción de código paralelo de un programa secuencial que, ejecutado en paralelo en 8 procesadores, tarda un tiempo de 100 ns, durante 50ns utiliza un único procesador y durante otros 50 ns utiliza 8 procesadores (distribuyendo la carga de trabajo por igual entre los procesadores)?

Solución

Datos del programa:



¿Fracción de código paralelizable del programa secuencial, f_p ?:



Ejercicio 4. Un 25% de un programa no se puede parallelizar, el resto se puede distribuir por igual entre cualquier número de procesadores. ¿Cuál es el máximo valor de ganancia de velocidad que se podría conseguir al parallelizarlo en p procesadores, y con infinitos? ¿A partir de cuál número de procesadores se podrían conseguir ganancias mayores o iguales que 2?

$$p \text{ procesadores} \rightarrow S(p) = \frac{1}{\frac{1}{8} + \frac{1-0.25}{p}} = \frac{1}{\frac{0.25}{0.25p+0.75}} = \frac{p}{0.25p+0.75}$$

$$\text{Infinitos procesadores} \rightarrow S(p) = \frac{p}{0.25p+0.75} \xrightarrow[p \rightarrow \infty]{} 4$$

$$S(p) = 2 \Leftrightarrow 2 = \frac{p}{0.25p+0.75} \Rightarrow 0.5p+1.5 = p \Rightarrow p = 3 \text{ procesadores ó más}$$

Ejercicio 5. En la Figura 1, se presenta el grafo de dependencia entre tareas para una aplicación. La figura muestra la fracción del tiempo de ejecución secuencial que la aplicación tarda en ejecutar las tareas del grafo. Suponiendo un tiempo de ejecución secuencial de 60 s, que las tareas no se pueden dividir en tareas de menor granularidad y que el tiempo de comunicación es despreciable, obtener el tiempo de ejecución en paralelo y la ganancia en velocidad en un computador con:

- (a) 4 procesadores.
- (b) 2 procesadores.

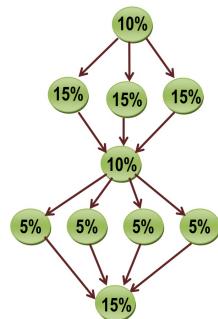
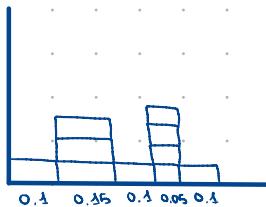


Figura 1. Grafo de tareas del ejercicio 5.

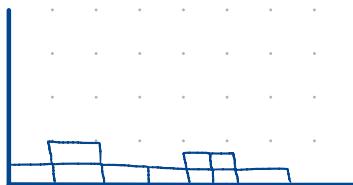
a) ¿ $T_p(4)$? ¿ $S(4)$? $T_s = 60$



$$T_p(4) = 0.1T_s + 0.15T_s + 0.1T_s + 0.05T_s + 0.15T_s = 0.55T_s = 33s$$

$$S(4) = \frac{T_s}{T_p(4)} = \frac{T_s}{0.55T_s} = 1.82 < 4$$

b)



$$T_p(2) = 0.1T_s + 2 \cdot 0.15T_s + 0.1T_s + 2 \cdot 0.05T_s + 0.15T_s = 0.75T_s = 45s$$

$$S(2) = \frac{T_s}{T_p(2)} = \frac{T_s}{0.75T_s} \approx 1.33 < 2$$

Ejercicio 6. Un programa se ha conseguido dividir en 10 tareas. El orden de precedencia entre las tareas se muestra con el grafo dirigido de la Figura 2. La ejecución de estas tareas en un procesador supone un tiempo de 2 sg. El 10% de ese tiempo es debido a la ejecución de la tarea 1; el 15% a la ejecución de la tarea 2; otro 15% a la ejecución de 3; cada tarea 4, 5, 6 o 7 supone el 9%; un 8% supone la tarea 8; la tarea 9 un 10%; por último, la tarea 10 supone un 6%. Se dispone de una arquitectura con 8 procesadores para ejecutar la aplicación. Consideramos que el tiempo de comunicación se puede despreciar. **(a)** ¿Qué tiempo tarda en ejecutarse el programa en paralelo? **(b)** ¿Qué ganancia en velocidad se obtiene con respecto a su ejecución secuencial?

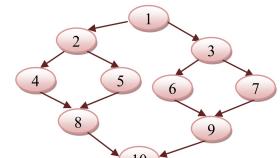
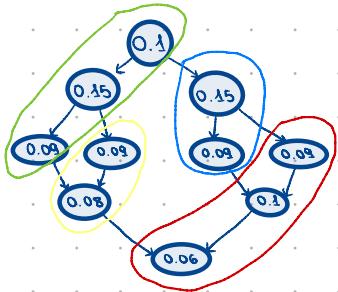


Figura 2. Grafo de dependencia entre tareas del ejercicio 6.

$$a) T_p(4) = 0.1T_s + 0.15T_s + 0.09T_s + 0.15T_s + 0.06T_s = 0.5T_s = 1 \text{ segundo}$$

$$b) S(4) = \frac{T_s}{T_p(4)} = \frac{T_s}{0.5T_s} = 2$$

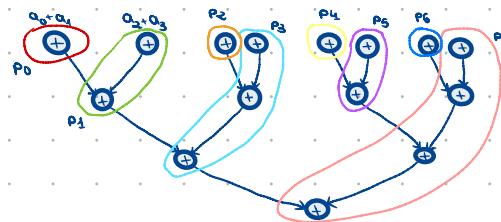
Ejercicio 8. Supongamos que se va a ejecutar en paralelo la suma de n números en una arquitectura con p procesadores o cores (p y n potencias de dos) utilizando un grafo de dependencias en forma de árbol (divide y vencerás) binario para las tareas.

- (a) Dibujar el grafo de dependencias entre tareas para $n = 16$ y $p = 8$. Hacer una asignación de tareas a procesos.
- (b) Obtener el tiempo de cálculo paralelo para cualquier n y p con $n > p$ suponiendo que se tarda una unidad de tiempo en realizar una suma.
- (c) Obtener el tiempo comunicación del algoritmo suponiendo (1) que las comunicaciones en un nivel del árbol se pueden realizar en paralelo en un número de unidades de tiempo igual al número de datos que recibe o envía un proceso en cada nivel del grafo de tareas (tenga en cuenta la asignación de tareas a procesos que ha considerado en el apartado (a)) y (2) que los procesadores que realizan las tareas de las hojas del árbol tienen acceso sin coste de comunicación a los datos que utilizan dichas tareas.
- (d) Suponiendo que el tiempo de sobrecarga coincide con el tiempo de comunicación calculado en (c), obtener la ganancia en prestaciones.
- (e) Obtener el número de procesadores para el que se obtiene la máxima ganancia con n números.

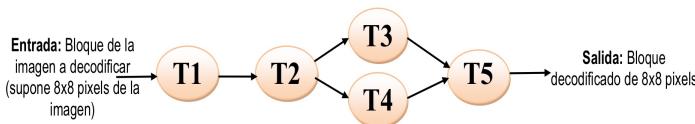
a) $n = 16 \quad p = 8 \quad \frac{n}{p} = 2$

$$a_0 \dots a_{15}$$

$$T_p(n,p) = \left(\frac{n}{p} - 1\right) + \log_2 p$$

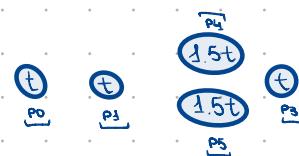


Ejercicio 9. Se va a parallelizar un decodificador JPEG en un multiprocesador. Se ha extraído para la aplicación el siguiente grafo de tareas que presenta una estructura segmentada (o de flujo de datos):



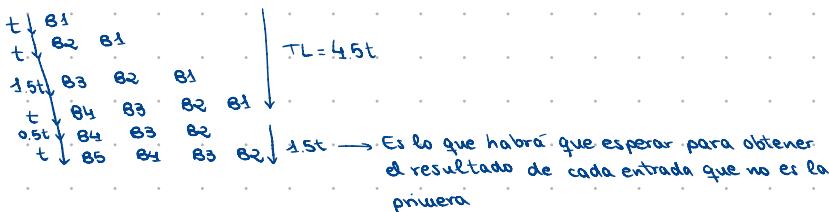
La tareas 1, 2 y 5 se ejecutan en un tiempo igual a t , mientras que las tareas 3 y 4 suponen 1,5t

El decodificador JPEG aplica el grafo de tareas de la figura a bloques de la imagen, cada uno de 8x8 píxeles. Si se procesa una imagen que se puede dividir en n bloques de 8x8 píxeles, a cada uno de esos n bloques se aplica el grafo de tareas de la figura. Obtenga la mayor ganancia en prestaciones que se puede conseguir paralelizando el decodificador JPEG en (suponga despreciable el tiempo de comunicación/sincronización): (a) 5 procesadores, y (b) 4 procesadores. En cualquier de los dos casos, la ganancia se tiene que calcular suponiendo que se procesa una imagen con un total de n bloques de 8x8 píxeles.



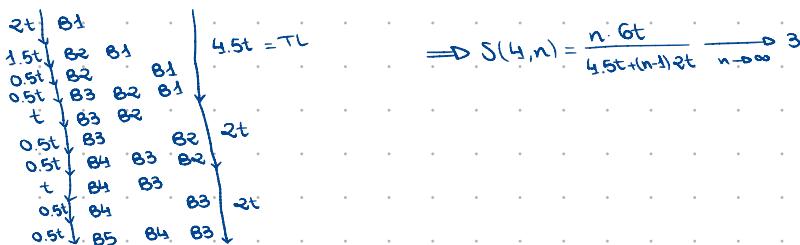
1^{er} resultado

$$a) T_p(5, n) = TL + (n-1)TPR = 4.5t + (n-1) \cdot 1.5t$$



$$S(5, n) = \frac{T_s(n)}{T_p(5, n)} = \frac{n \times 6t}{3t + (n-1) \cdot 1.5t} \xrightarrow{n \rightarrow \infty} \frac{6}{1.5} = 4$$

b) Uno se encarga de las 2 primeras etapas, y luego el resto de una etapa:



Cuestión 1. Indique las diferencias entre OpenMP y MPI.

- OpenMP → Usa variables compartidas?
 - MPI → Usa el paso de mensajes
 - API basada en directivas de compilador y funciones
 - API basada en funciones de biblioteca (menor abstracción que OpenMP).
- Estandares de facto para la programación paralela

Cuestión 2. Ventajas e inconvenientes de una asignación estática de tareas a procesos/threads frente a una asignación dinámica.

-) Ventajas estática frente a dinámica:
 - Menos instr. extra que la dinámica.
 - No hay comunic./ sincr. para asignar tareas a flujos durante la ejecución.
- .) Ventajas dinámica frente a estática:
 - Se puede usar en aplicaciones en las que no se sabe con seguridad el nº de tareas a ejecutar.
 - Permite equilibrio de carga cuando la plataforma es heterogénea y/o no uniforme.
 - Permite equilibrio de carga cuando las tareas a repartir entre flujos requieren distinto tiempo.

Cuestión 3. ¿Qué se entiende por escalabilidad lineal y por escalabilidad superlineal? Indique las causas por las que se puede obtener una escalabilidad superlineal.

Escalabilidad lineal: Se da cuando la ganancia en prestaciones es igual al nº de recursos utilizados. Representa la escalabilidad lineal que se esperaría conseguir.

Escalabilidad superlineal: Está por encima de la lineal. Esto se debe a que en la práctica se añaden varios recursos de distintos tipos, y si la aplicación se beneficia de más de uno de estos tipos de recursos, la ganancia puede resultar por encima de la lineal.

Cuestión 4. Enuncie la ley de Amdahl en el contexto de procesamiento paralelo.

La ganancia en prestaciones que se puede conseguir al añadir procesadores está limitada por la fracción del tiempo de ejecución secuencial que supone la parte del código no parallelizable.

Cuestión 5. Deduzca la expresión matemática que se suele utilizar para caracterizar la ley de Gustafson. Defina claramente y sin ambigüedad el punto de partida que va a utilizar para deducir esta expresión y cada una de las etiquetas que utilice. ¿Qué nos quiere decir Gustafson con esta ley?

Partimos de un código en el que T_s no permanece cte. al variar el nº de procesadores, lo que permanece constante es T_p , en el que hay un trozo no parallelizable que supone una fracción de tiempo f y otro trozo parallelizable que se reparte por igual entre los procesadores disponibles. La ganancia en prestaciones (suponiendo $T_0=0$) viene dada por la expresión:

$$S(p) = \frac{T_s}{T_p} = \frac{f \cdot T_p + (1-f) \cdot T_p \cdot p}{T_p} = f + (1-f)p$$

Cuestión 6. Deduzca la expresión que caracteriza a la ley de Amdahl. Defina claramente el punto de partida y todas las etiquetas que utilice.

Se parte de un código secuencial que tiene una parte no parallelizable que supone una fracción de tiempo f que permanece cte. a pesar de incrementar el nº de procesadores. El resto se puede parallelizar de forma ilimitada, repartiéndose por igual entre los p procesadores disponibles. La ganancia en prestaciones ($T_0=0$) viene dada por la expresión:

$$S(p) = \frac{T_s}{T_p(p)} = \frac{T_s}{\frac{f \cdot T_s + (1-f) \cdot T_s}{p}} = \frac{p}{1 + f(p-1)}$$