Facultad de Informática

UPV/EHU

eman ta zabal zazu

Universidad del País Vasco Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Arquitectura de **C**omputadores

Departamento de Arquitectura y Tecnología de Computadores

Ejercicios Paralelismo

3.1. Un multiprocesador tiene 64 núcleos. Se ha analizado un programa que va a ejecutarse en esa máquina y se han obtenido los siguientes datos: una parte del programa, 92%, se puede paralelizar completamente; otra parte, 6%, se puede ejecutar sólo en 32 procesadores; el resto debe ejecutarse en serie. Calcula el factor de aceleración (*speed-up*) y la eficiencia que se conseguirán (*efficiency*).

3.2. Cuando se utilizan *P* núcleos para ejecutar aplicaciones en paralelo el objetivo es conseguir tiempos de ejecución *P* veces más pequeños (esto es, que el factor de aceleración sea *P*). Sin embargo, hay varios factores que limitan ese rendimiento teórico.

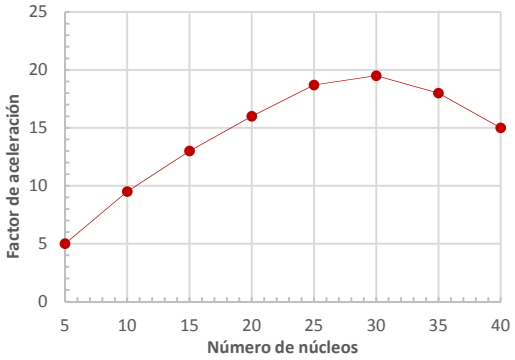
a. En una determinada aplicación, un 2% no se puede ejecutar en paralelo. Calcula el factor de aceleración máximo que se puede conseguir en el límite (utilizando un número muy elevado de procesadores).

b. Una aplicación se puede paralelizar por completo, pero surgen varias situaciones de sobrecarga (sincronización, equilibrio en el reparto de trabajo...). Por ello, el tiempo de ejecución del proceso paralelo se incrementa en un 20%. ¿Cuál será la eficiencia máxima que se puede conseguir en esa aplicación en un máquina con *P* núcleos?

c. Teniendo en cuenta los dos problemas anteriores, calcula el factor de aceleración y la eficiencia que conseguirán en una máquina con 32 núcleos.

3.3. Se ejecuta un programa paralelo en una máquina SMP con 8 núcleos. A lo largo de la ejecución, se pueden utilizar los 8 núcleos durante 50 segundos; 4 núcleos durante 20 segundos; y un único núcleo durante 30 segundos. La carga de trabajo se reparte de forma equilibrada entre todos los procesos, pero en el cálculo paralelo se genera una sobrecarga (*overhead*) de un 10%. Calcula el factor de aceleración (*speed-up*) y la eficiencia (*efficiency*) conseguidos.

3.4. Se ha ejecutado una aplicación en una máquina SMP, y se consigue el factor de aceleración de la figura. ¿Qué número de núcleos hay que utilizar para conseguir una eficiencia del 80%? Justifica tu respuesta.



3.5. En un determinado programa se procesan vectores de *N* elementos. Una parte del código no puede ser paralelizada y su tiempo de ejecución es de 500 ms. El resto de código puede paralelizarse sin problemas y su tiempo de ejecución serie es proporcional al tamaño del vector: 100 *N* ms.

Queremos ejecutar una versión paralela del programa en una máquina SMP de 10 núcleos para un vector de tamaño $N = 100$. ¿Cuál es la fracción *f* para esa ejecución? ¿Qué factor de aceleración y eficiencia se conseguirán?

3.6. El tiempo de ejecución de una aplicación paralela es proporcional al tamaño de los vectores que se procesan: $20 \times N$ ms. El código se puede ejecutar en paralelo de esta forma: cada proceso procesará su correspondiente trozo del vector, calculando un valor parcial; en una segunda fase, deben procesarse los valores parciales calculados para obtener el resultado definitivo. El tiempo de ejecución de la segunda fase es proporcional al número de núcleos utilizados: $10 \times P$ ms.

a. Calcula el factor de aceleración y la eficiencia que se consiguen si se procesa un vector de 200 elementos en una máquina con 10 núcleos.

b. Calcula el factor de aceleración máximo que se puede conseguir con dicho código (tiempo de ejecución paralelo mínimo) y cuántos núcleos deben utilizarse para conseguirlo. Grafica el factor de aceleración en función de *P* (*P* = 2, 4, 8, 16, 32 eta 64), para estos tamaños de vector, *N* = 200, 400, 600 y 800.

3.7. Se ejecuta una determinada aplicación en paralelo. El cálculo que debe realizarse, T_{calc}, se puede repartir de forma equilibrada entre todos los procesos, pero la ejecución paralela genera una sobrecarga (generación de hilos, sincronización…) que puede expresarse de esta forma:

T_{sob}= 800 + 50 x *P* milisegundos.

a. Calcula la eficiencia que se consigue en la ejecución paralela si T_{calc}=200 segundos y P=20 procesos.

b. Calcula, en función de P, cuál es el tiempo de cálculo mínimo (ms) del programa paralelo para que la eficiencia sea como mínimo del 75%. Indica este tiempo para P=20 procesos.

3.8. Parte de una aplicación paralela puede paralelizarse totalmente, pero otra parte debe ejecutarse en serie. Se utiliza una máquina de 64 núcleos para ejecutar esa aplicación.

a. La aplicación se comporta de acuerdo a la ley de Amdahl. Calcula la fracción de código mínima que debe ejecutarse en paralelo, *f*, para conseguir una eficiencia del 50%. ¿Qué factor de aceleración se consigue si *f* = 0,9?

b. La aplicación se comporta de acuerdo a la ley de Gustafson. Calcula el factor de aceleración que se consigue si *f* = 0,9.

3.9. En la ejecución de una aplicación paralela se distinguen dos partes: una parte de E/S que debe ejecutarse en serie y otra parte, un código iterativo, que puede paralelizarse completamente. La parte serie tiene un tiempo de ejecución de 100 s, mientras que la segunda parte requiere de 50 s por cada iteración.

Hemos ejecutado el programa de forma paralela en una máquina con 16 núcleos. Calcula el factor de aceleración y la eficiencia que se consiguen si se ejecutan 16 iteraciones en el programa. ¿Y si se ejecutaran 1.600 iteraciones? ¿Qué ley gobierna el comportamiento del programa?

3.10. Se ejecutan 100 iteraciones independientes en una máquina paralela de 4 procesadores. Cada iteración tiene un tiempo de ejecución de 1 s, salvo una de ellas que requiere de un tiempo de ejecución de 20 s. Calcula el factor de aceleración y la eficiencia que se consiguen en estos casos:

a. El reparto de iteraciones es estático, consecutivo (trozos del mismo tamaño).

b. El reparto de iteraciones es estático, entrelazado (de uno en uno).

c. El reparto de iteraciones es dinámico, de uno en uno. Calcula el peor caso y el mejor caso. La asignación dinámica de iteraciones genera una sobrecarga de 50 ms por asignación.

3.11. Una máquina SMP tiene 4 procesadores. En esa máquina se ejecutan 8 tareas independientes. El tiempo de ejecución (segundos) de cara tarea es el siguiente: 5, 10, 6, 15, 8, 3, 2 y 1. Calcula el factor de aceleración y la eficiencia que se consiguen en estos dos casos: **(a)** reparto estático consecutivo (`static`) y **(b)** reparto dinámico, de uno en uno (`dynamic, 1`).

3.12. En estos dos trozos de código se procesa de forma paralela una matriz `A[10][5]`. El reparto de iteraciones es estático consecutivo.

a.
```
#pragma omp parallel for private (i, j) schedule (static)
for (i=0; i<10; i++)
  for (j=0; j<5; j++)
    A[i][j] = A[i][j] * i * j;
...
```

b.
```
for (i=0; i<10; i++)
  #pragma omp parallel for private (j) schedule (static)
  for (j=0; j<5; j++)
    A[i][j] = A[i][j] * i * j;
...
```

Vamos a ejecutar los dos códigos en una máquina con 5 núcleos. Indica qué elementos de la matriz `A` procesará en cada caso el hilo `pid` = 2. Razona tu respuesta.

3.13. Se quiere ejecutar el siguiente código de forma paralela utilizando OpenMP:

```
pragma omp parallel private (i)
{
  ...
  #pragma omp for
  for (i=0; i<N; i++)
    if (A[i] == 5) kodetu (i, A);
  ...
  #pragma omp for
  for (i=0; i<N; i++)
    B[i] = B[i]*B[i] / A[i]*A[i];
  ...
}
```

Completa la directiva `for`, indicando la cláusula de planificación que más adecuada consideres. Razona tu respuesta.

3.14. En OpenMP las directivas `parallel` y `for` tienen una barrera implícita al final para sincronizar todos los hilos. En algunos casos, esa barrera se puede eliminar. En este ejemplo, tenemos 3 bucles `for`. Indica si la barrera se puede eliminar en alguno de los casos, por qué y cómo.

```
#pragma omp parallel private (i)
{
  #pragma omp for schedule (dynamic, 1)
   for (i...) A[i] = f1(A[i]);
  #pragma omp for schedule (dynamic, 1)
   for (i...) B[2*i] = f2(B[2*i]);
  #pragma omp for schedule (dynamic, 1)
   for (i...) C[3*i] = A[3*i] + B[3*i];
}
```

3.15. Se quiere ejecutar este trozo de código en un multiprocesador. Escribe el código paralelo utilizando OpenMP para estos dos casos: (a) utilizando una variable de tipo `reduction`; (b) sin utilizar variables de ese tipo.

```
Amin = INT_MAX;
for (i=0; i<N; i++) if (A[i] < Amin) Amin = A[i];
for (i=0; i<N; i++) A[i] = A[i] - Amin;
```

3.16. Se quiere ejecutar este trozo de código en un multiprocesador. Escribe el código paralelo utilizando OpenMP.

```
batura = 0.0;
for (i=0; i<N; i++) batura += + A[i]*A[i];
bb = batura / N;
for (i=0; i<N; i++) A[i] = sqrt (A[i] + bb) + pow (bb, 4);
```

3.17. El siguiente programa calcula el histograma de una variable, esto es, cuántos píxeles de un determinado valor existen en la imagen. Escribe el código paralelo utilizando OpenMP.

```
...
typedef struct {
  int  h, w;
  unsigned char  pix[N][M];
} imagen;

main ( )
{
  int  i, j, histo[256];
  struct imagen  im1;
  for (i=0; i<256; i++) histo[i] = 0;
  for (i=0; i<im1.h; i++)
  for (j=0; j<im1.w; j++)
    histo[im1.pix[i][j]] ++;
  ...
}
```

3.18.

1. Utiliza una hoja de cálculo (por ejemplo, excel) y realiza un gráfico que muestre el comportamiento de la ley de Amdhal: factor de aceleración que se consigue en una aplicación paralela en función del número de procesos, para distintas fracciones de código paralelizable. Por ejemplo, realiza un gráfico para el intervalo de $P = 2..128$ y para valores de $f = 0{,}7,\ 0{,}8,\ 0{,}9,\ 0{,}95$ y 1 (caso ideal).

2. Al ejecutar una aplicación en paralelo entre P procesadores se quiere conseguir una eficiencia superior a 0,5. Demuestra que la fracción a ejecutar en serie tiene que ser inferior a $1 / (P - 1)$.

3. De acuerdo a la clasificación de Flynn, los computadores pueden clasificarse en estas tres categorías: sistemas SISD, sistemas SIMD y sistemas MIMD, en este último caso se contemplan dos subcategorías: memoria compartida y memoria privada.

Acabas de comprarte un portátil con estas características: un procesador con 4 núcleos, 16 GB de RAM, 128 MB de cache (en dos niveles) y 1 TB de disco duro. ¿En qué categoría se encuentra tu máquina? Razona la respuesta.

4. Se ejecuta este código en paralelo en una máquina con 100 núcleos. Se trata de tres bucles, por lo que existen varias opciones de paralelización. ¿Cuál será la más eficiente? Justifica tu respuesta. Para la opción que hayas decidido completa el programa paralelo con los `pragma` necesarios. Indica también la planificación más adecuada y haz una hipótesis sobre el factor de aceleración que se conseguirá.

```
for (i=0; i<50; i++)
{
  for (j=0; j<400; j++)
  for (k=0; k<5; k++) {
    x = fun (A[i][j][k]);
    C[i] += x;
  }
}
```

5. Dos cuestiones teóricas. (a) ¿Qué es una barrera en un sistema SMP paralelo? ¿Para qué se utiliza? Pon un ejemplo de un trozo de programa en el que sea necesario utilizar una barrera (utiliza la sintaxis de OpenMP). (b) ¿Qué quiere decir que una operación sea atómica? Pon un ejemplo.

6. Queremos ejecutar en paralelo este trozo de código:

```
for (i=0; i<N; i++)  x += C[i+1];
for (j=0; j<M; j++)  B[j] ++;
for (j=0; j<M; j++)  B[j] = B[j] * x;
```

Para ejecutarlo de forma correcta son necesarias una sección crítica y una barrera de sincronización. Explica dónde son necesarias y por qué hay que utilizarlas.

7. Queremos ejecutar 100 iteraciones de un bucle en paralelo entre 10 procesos. Indica qué iteraciones ejecutará el proceso P5 en estos dos casos: (1) reparto estático, de dos en dos; (2) reparto dinámico, también de dos en dos.

8. Un programa tiene tres funciones independientes, con estos tiempos de ejecución: 10, 12 y 15 minutos. Las tres funciones se pueden ejecutar en paralelo utilizando el `pragma sections` de OpenMP (las funciones se reparten entre los núcleos). Calcula el factor de aceleración y la eficiencia que se conseguirán al ejecutar en paralelo el programa entre 2 procesos y entre 3 procesos.

9. Se ha ejecutado un bucle de 16 iteraciones en paralelo entre 4 procesos de forma que las iteraciones se han repartido así:

a. P0: 0, 1, 2, 3 P1: 7, 8, 9, 12, 14 P2: 4, 5, 6, 13, 15 P3: 10, 11

b. P0: 0, 1, 2, 3 P1: 4, 5, 6, 7 P2: 8, 9, 10, 11 P3: 12, 13, 14, 15

c. P0: 0, 1, 8, 9, 14, 15 P1: 6, 7, 10, 11 P2: 4, 5, 12, 13 P3: 2, 3

Razona cuál ha sido el reparto de trabajo (*scheduling*) en cada caso.

10. Razona si las siguientes sentencias son verdaderas o falsas y por qué (no es suficiente indicar si son verdaderas o falsas)

- Es suficiente sincronizar las lecturas de las variables compartidas. Una vez sincronizadas las lecturas, las escrituras se realizarán en orden.
- Una barrera de sincronización no supone una pérdida de rendimiento, dado que, una vez sincronizados los procesos, estos reinician la ejecución paralela.
- El código de una sección crítica la ejecuta un sólo proceso, el primero que llegue a ese punto.
- Estas tres operaciones son operaciones de reducción:

```
...                        for i...              for i...
if (x < min) min = x;        k = k + A[i];         B[i] = B[i] * 2;
...
```

- Para realizar un reparto estático hay que crear una sección crítica. En caso de que la planificación sea dinámica (*self-scheduling*), la sección crítica no es necesaria, dado que cada proceso elegirá que debe ejecutar.
- El tiempo de ejecución de una sección crítica supone un 5% del tiempo de ejecución de un programa paralelo. En el mejor caso, se tiempo no influye en la fracción de programa que debe ejecutarse en serie, pero en el peor de los casos sí influye.