ÍNDICE

Introducción

Este documento recoge preguntas de exámenes CEX (teoría) de la asignatura PCP de cursos pasados. Debe tenerse en consideración que la materia, así como su distribución en las diferentes pruebas de evaluación CEX, ha cambiado.

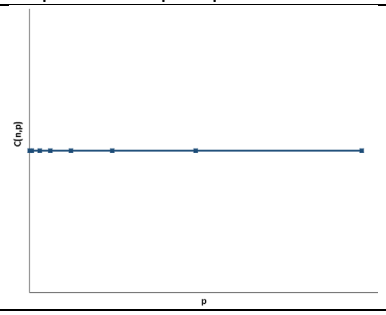
El documento está dividido en secciones, conteniendo cada una de ellas preguntas relacionadas con la materia objeto de la sección (tema).

Los prefijos **CEX1** y **CEX2** en los títulos de sección indican si las preguntas corresponden al primer o al segundo examen CEX de PCP, respectivamente.

Los problemas no están divididos en secciones. No obstante, están en orden inverso (primero los de cursos más "antiguos") y los correspondientes a cursos más recientes está etiquetados con el año y la fecha (p.ej. "Curso académico 18/19 – octubre 2018"). De esta forma es sencillo determinar si corresponden a **CEX1** o **CEX2**.

CEX 1: Preguntas de Modelos de Rendimiento

Pregunta. Sea la siguiente grafica del coste para un valor de n concreto y fijo. ¿Cómo calificaría el algoritmo paralelo? ¿por qué?



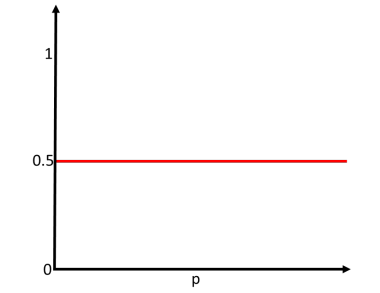
Respuesta
Se puede calificar como "óptimo". Para un tamaño n fijo al aumentar el valor de p el tiempo de ejecución paralelo disminuye proporcional, manteniendo el coste constante. Más formalmente:

$$\lim_{\substack{p\to\infty \\ n\ fijo}} C(n,p) = cte \Leftrightarrow cteT(n,p) = \frac{T(n)}{p}$$

Pregunta. ¿Qué es más eficiente $(t_s + nt_w)$ o $n(t_s + t_w)$ ¿Por qué?

Respuesta
Pregunta clásica de teoría. Está en los apuntes. $(t_s + nt_w)$ es más eficiente.

Pregunta. Sea $T(n) = \frac{n^3 t_c}{3}$ y $T(n,p) = \frac{2n^3 t_c}{p\sqrt{9}}$. Dibuje, con precisión, exactitud y sin obviar información relevante la eficiencia escalada.



Respuesta
La eficiencia es constante (0.5) independientemente del valor de p.

$$E(n,p) = \frac{\frac{n^3 t_c}{3}}{p\frac{2n^3 t_c}{p\sqrt{9}}} = \frac{\frac{n^3 t_c}{3}}{\frac{2n^3 t_c}{3}} = \frac{1}{2}$$

Pregunta. Sea $T(n) = (n^4 + n^3)t_c$ y $T(n,p) = (2n^3 + n^4/p)t_c$. Según Gustafson-Barsis ¿el diseño paralelo es adecuado, o no, para máquinas masivamente paralelas? Demostrar la afirmación.

Respuesta

$$E(n,p) = \frac{n^4 + n^3}{p(2n^3 + \frac{n^4}{p})} = \frac{n^3(n+1)}{n^3(n+2p)} = \frac{(n+1)}{(n+2p)} = c \Rightarrow (n+2p)c = (n+1)$$

$$cn + 2cp = n + 1; 2cp - 1 = n - cn = n(1-c); n = \frac{2cp-1}{1-c}$$

El algoritmo es apropiado para máquinas masivamente paralelas puesto que un crecimiento lineal en n necesita un crecimiento lineal en p

Pregunta. ¿Significa lo mismo $T(n,1)$ que $T(n)$? De no ser así, ¿Cuándo se debe usar cada expresión?

Respuesta
No $T(n,1)$ denota la ejecución del algoritmo paralelos sobre un solo procesador y $T(n)$ representa el tiempo de ejecución del algoritmo secuencial óptimo (ver teoría).

Pregunta. Sea un algoritmo secuencial tal que $T(n) = n/4$ y otro paralelo tal que $T(n,p) = \frac{n\sqrt{9}}{p\sqrt[3]{27}}$. ¿Qué valores de n permiten obtener una eficiencia 0.5? Las conclusiones a las que ha llegado ¿Qué le sugieren?

Respuesta
Ningún valor de n. La eficiencia es constate con valor 0.25. $C(n,p) = n$ y $T_0(n,p) = \frac{3}{4}n$. Que el comportamiento (rendimiento) del algoritmo paralelo no depende del número de procesadores, solo depende del tamaño del problema.

Pregunta. Sea un algoritmo secuencial tal que $T(n) = n/4$ y otro paralelo tal que $T(n,p) = \frac{n\sqrt{9}}{p\sqrt[3]{27}}$. ¿Qué valores de n permiten obtener una eficiencia 0.25? Las conclusiones a las que ha llegado ¿Qué le sugieren?

Respuesta
Cualquier valor de n. Iguales reflexiones que en la pregunta anterior.

Pregunta. En el análisis de la complejidad temporal de un algoritmo se obtiene la siguiente expresión en lo referente a las comunicaciones: $\left(\frac{n^2}{p}t_s + \frac{n^2-p}{p}t_w\right)$ ¿es factible? Razone la respuesta.

Respuesta
El término multiplicativo que acompaña a t_w es el número de elementos enviados y el que multiplica a $t_s$el número de operaciones de comunicación realizadas. Que el factor multiplicativo de t_w sea menor solo es posible si se envían p mensajes de tamaño 0 (o un número superior compensado por el tamaño > 1 de otros mensaje). En el contexto de esta asignatura no es factible porque las operaciones de sincronización (o similares) se modelizan mediante mensajes de tamaño constante ≥ 1.

Pregunta. Sea $T(n) = 2nt_c$ y el *speedup* $0.5p$. ¿Cuál sería una expresión válida de $T(n,p)$?

Respuesta
$0.5p = \frac{2nt_c}{T(n,p)} \Longrightarrow T(n,p) = \frac{2nt_c}{0.5p} = \frac{4nt_c}{p}$, o cualquier otra expresión equivalente.

Pregunta. Sea $T_0(n,p) = p(t_s + nt_w)$ y $W = n^2$. Realizar el análisis de la isoeficiencia.

Respuesta
$t_s: n^2 = W \propto Kp \in \theta(p)$
$t_w: n^2 = W \propto Knp = KW^{\frac{1}{2}}p;\ W^{\frac{1}{2}} \propto Kp; W \propto Kp^2\ \theta(p^2)$
$max\big(\theta(p), \theta(p^2)\big)$=$\theta(p^2) \Rightarrow I(W,p) = \frac{\theta(p^2)}{\theta(n^2)}$

Pregunta. En los apuntes de teoría se dice que, en general, la cota de complejidad temporal para una operación colectiva básica entre p procesos se pueden aproximar por $p(t_s + nt_w)$, siendo n el tamaño del mensaje. Sea un vector de tamaño n que se difunde desde el procesador raíz de una topología tipo árbol. ¿Se podría resolver el problema de la difusión con una cota de complejidad inferior a la mencionada? En caso afirmativo, esboce la expresión analítica de dicha cota.

Respuesta
Sí, estaría acotada por $k(t_s + nt_w)\log_k(p)$, siendo un árbol *k-ario*

Pregunta. Y si fuese una distribución ¿la cota sería la misma o similar? ¿Por qué?

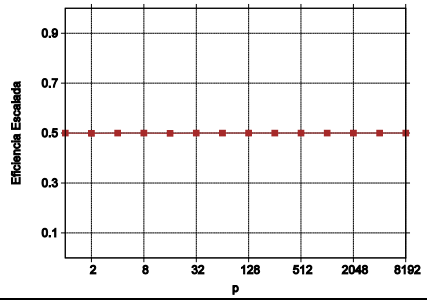
Respuesta
Sí, solo cambia el tamaño de mensaje que a medida que se recorre el árbol disminuye. Analíticamente (no se pide) sería $\approx k\sum_{i=1}^{\log_k(p)}\left(t_s + \left(\frac{n}{k^i} - \frac{n}{p}\right)t_w\right)$, siendo un árbol *k-ario*

Pregunta. Sea un algoritmo secuencial tal que $T(n) = n\ /\ 4$ y otro paralelo tal que $T(n,p) = 2n/8p$ ¿Depende la escalabilidad de p? ¿Qué significado se le debe dar al valor de la función obtenido en el análisis de la escalabilidad? ¿Se podrían haber sabido las respuestas a las preguntas formuladas sin realizar cálculos? Razone las respuestas.

Respuesta

No. Dado que $T(n,p)$ es $T(n)$ dividido por p (óptimo) la eficiencia es 1 y constante. Sí por lo dicho, eficiencia 1 constante que implica coste 0 y función de sobrecarga 0.

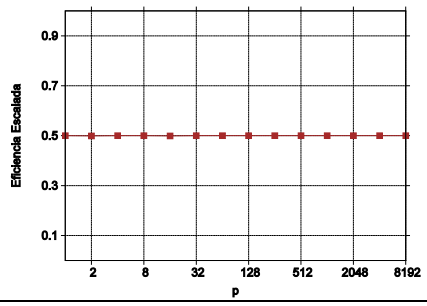
Pregunta. Sea $W = n^2$, $T(n) = n^2 t_c$, p el número de procesadores y la gráfica de la *Eficiencia Escalada* adjunta. ¿Cuál es la expresión del tiempo de ejecución paralelo?



Respuesta

$$T(n,p) = \frac{2n^2 t_c}{p}$$

Pregunta. Sea la *Eficiencia Escalada* resultante de aumentar proporcionalmente W y p (p es el número de procesadores) mostrada en la gráfica adjunta. Escriba expresiones de W, tiempo de ejecución secuencial y paralelo coherentes con la gráfica.



Respuesta

$$T(n,p) = \frac{2n^2 t_c}{p}$$

$$T(n) = n^2 t_c$$

$$W = n^2$$

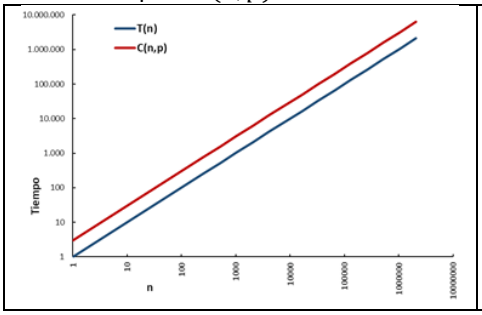
Pregunta. Sea $T(n) = 2n^2 t_c$ y $T_o(n,p) = -n^2 t_c$; no hay comunicaciones. ¿Qué está pasando? ¿Puede argumentarlo y/o demostrarlo?

Respuesta

$$-n^2 t_c = T_o(n,p) = pT(n,p) - 2n^2 t_c \Rightarrow pT(n,p) = n^2 t_c \Rightarrow T(n,p) = \frac{n^2 t_c}{p}$$

Que la eficiencia es 2, mayor que 1. Por tanto, error de cálculo/formulación o el secuencial de referencia no es el óptimo. Definir esto como súper-linealidad no sería adecuado porque no es un efecto colateral, es una expresión analítica.

Pregunta. Sea la gráfica adjunta de $C(n,p)$ y $T(n)$, válida para cualquiera valor de p. Presentar una ecuación válida para $T(n,p)$. ¿Cómo calificaría el algoritmo paralelo? ¿Por qué?



Respuesta

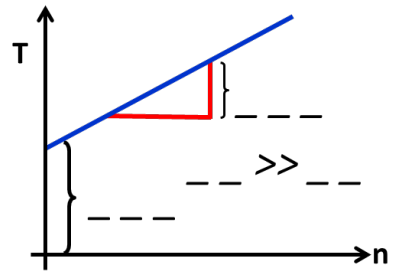
$pT(n,p) = C(n,p) = cteT(n) \Rightarrow T(n,p) = \frac{cteT(n)}{p}$

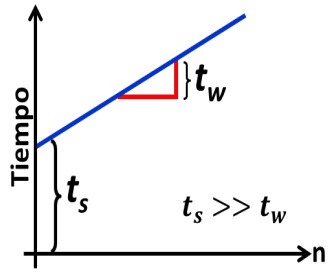
Óptimo porque $C(n,p)$ es proporcional a $T(n)$

Pregunta. En un problema de tamaño n fijo el $T(n,p)$ empieza, en un momento dado, a crecer. ¿Cuándo y por qué?

Respuesta
Por la ley de Amdahl cuando p supera un determinado umbral el tiempo paralelo puede crecer.

Pregunta. Complete la siguiente gráfica.





Respuesta

Pregunta. Sea $T(n) = n^2 t_c$ y $T(n,p) = \frac{n^2}{p} t_c + (t_s + pn^2 t_w)$. Sin hacer cálculos ¿es escalable? Razone la respuesta.

Respuesta
NO. El algoritmo paralelo añade comunicaciones cuyo coste, que depende de W y de p, es de orden superior a W.

(Curso 2019 – 2020)

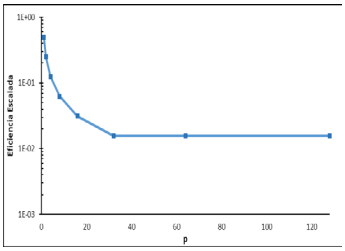
Pregunta. Sea la siguiente afirmación: "*Para que un algoritmo sea escalable la eficiencia tiene que ser 1*". Si es correcta responda **SÍ**. Si no lo es, reescriba la frase con el menor número de cambios, sin alterar la esencia de la afirmación.

Respuesta
NO es correcta. Sin alterar la esencia y con el menor número de cambios podría ser:
"*Para que un algoritmo sea escalable la eficiencia tiene que ser constante*".

Pregunta. Cuando la Fracción Paralela (PF) es muy baja un procesador con muchos *cores* tendrá un Rendimiento Efectivo (EF) peor que con 2 cores. ¿Cierto o falso? ¿por qué?

Respuesta
Falso. Solo cuando PF es del 0% las EF se igualan. Para PFs pequeñas la diferencia entre los EFs se reduce, pero teóricamente no se cruzan.

Pregunta. Sea la siguiente de la eficiencia escalada. Si se desea usar 1024 procesadores ¿qué se podría extrapolar observando la figura?



Respuesta
Con seguridad no se puede extrapolar nada. Habría que estudiar el comportamiento del algoritmo para un rango de procesadores mayor que el de la figura (que incluya 1024 en el análisis). Posiblemente el comportamiento sería el mismo que el observado para 64 o 128 procesadores, pero no se puede asegurar.

Pregunta. En un foro de internet aparece la siguiente definición: $E(n,p) = T(n)/C(n,p)$. ¿Es correcta?

Respuesta
$E(n,p) = \frac{T(n)}{pT(n.p)}$. Por otra parte $C(n,p) = pT(n,p)$. Por tanto, sí, es correcta.

Pregunta. Se dispone de un algoritmo paralelo de $E(n,p) = 0.4$. El algoritmo secuencial, que tarda 100 sec. en resolver el problema, no responde en tiempo real (en menos de 10 sec.). ¿Se podría garantizar que el paralelo lo resuelve en tiempo real? Razone la respuesta y, si es afirmativa, dar una estimación del número de procesadores necesarios.

Respuesta
$T(n) = pE(n,p)T(n,p)$; como $T(n) = 100$ y $E(n,p) = 0.4$ se tiene que $T(n,p) = \frac{250}{p}$. Como se desea que $T(n,p) < 10$ entonces para que se cumpla $\frac{250}{p} < 10$ que al ser $p \in \mathbb{N} \Longrightarrow p \geq 26$ cumple el requerimiento de tiempo real.

(Curso 2020 – 2021)

Pregunta. Un programador ha implementado un algoritmo y afirma que "$T(n)$ es constante y que $T(n,p)$ *disminuye* proporcionalmente *al aumentar* p". ¿Qué está diciendo implícitamente de 2 de los parámetros relativos más usados en el análisis del rendimiento? Razonar la respuesta (modelos A y B).

Respuesta
Si $T(n)$ es constante y $T(n,p)$ disminuye proporcionalmente al aumentar p está diciendo que la eficiencia es constante $\left(E(n,p) = \frac{T(n)}{pT(n,p)}\right)$ y, por tanto, que el algoritmo es escalable. Si la discusión es sobre el coste la conclusión es la misma, $C(n,p)$ es constante, lo mismo que la sobrecarga $\left(T_0(n,p) = C(n,p) - T(n)\right)$ y respecto al $S(n,p)$ $\left(\frac{T(n)}{T(n,p)} = pE(n,p)\right)$ que crece proporcionalmente con p.

Pregunta. El TPP_{dp} de un ordenador con un procesador de 4 núcleos es 100 Gflop. ¿Qué valor se estima adquiere t_c en $T(n) = nt_c$? Razonar la respuesta (modelo A).

Respuesta
Si el TPP_{dp} de un ordenador con un procesador de 4 núcleos es $100\ Gflop$ entonces el $t_c$del ordenador, cuando se usan todos los núcleos, es $1/1x10^{11} = 1x10^{-11}$ segundos. Como preguntan por la variante secuencial ($T(n)$) el valor esperable es t_c es $4x10^{-11}$ segundos.

(apartado 1º de la Práctica 1ª)

Pregunta. El TPP_{dp} de un ordenador con un procesador de 2 núcleos es 100 Gflop. ¿Qué valor se estima adquiere t_c en $T(n) = nt_c$? Razonar la respuesta (modelo B).

Respuesta
Si el TPP_{dp} de un ordenador con un procesador de 2 núcleos es $100\ Gflop$ entonces el t_c del ordenador, cuando se usan todos los núcleos, es $1/1x10^{11} = 1x10^{-11}$ segundos. Como preguntan por la variante secuencial ($T(n)$) el valor esperable es t_c es $2x10^{-11}$ segundos.
(apartado 1º de la Práctica 1ª)

Pregunta. **Complete**, con precisión y exactitud, la siguiente frase relativa a la escalabilidad (modelo A).

Respuesta
La $I(W,p)$ de un algoritmo es $O(pn^2)/O(n^2)$ por lo que el algoritmo **no es escalable**, mientras que otro con $I(W,p) = O(p)/O(\sqrt{n})$ **tampoco es escalable**. En ambos casos el orden del numerador es mayor que el del denominador.
(resuelta en clase durante la explicación de un problema)

Pregunta. **Complete**, con precisión y exactitud, la siguiente frase relativa a la escalabilidad (modelo B).

Respuesta
La $I(W,p)$ de un algoritmo es $O(p)/O(n^2)$ por lo que el algoritmo **es escalable**, mientras que otro con $I(W,p) = O(p)/O(\sqrt{n})$ **no es escalable**.
(resuelta en clase durante la explicación de un problema)

CEX 2: Preguntas de OpenMP

Pregunta. Sea un vector V, de tamaño n, tal que a todos y cada uno de sus elementos se le debe aplicar una función $f(V[i]) \in \theta(n)$. La solución debe ser apta para cualquier ordenador multi-núcleo. De todos los *Worksharing Constructs* "*usados*" ¿cuál/cuáles no se debería usar desde la perspectiva de la adaptabilidad? ¿Por qué?

Respuesta
No se debería usar *sections* porque cada vez que el número de núcleos del ordenador usado no coincidiese con lo codificado habría que modificar el programa.

Pregunta. Sea el siguiente código. ¿Qué imprime? Razone la respuesta.

```
#include <omp.h>
int main() {
  int x=0;
  #pragma omp parallel sections
  {
    #pragma omp single
      #pragma omp parallel
        x++;
  }
  printf("%d\n", x);
  return 0;
}
```

Respuesta
No imprime nada. Simplemente es incorrecto: "*…error: work-sharing region may not be closely nested inside of work-sharing...*"

Pregunta recurrente en los exámenes.

Pregunta. Sea el código adjunto. ¿Cuántos hilos se despliegan? Razone la respuesta.

```
#include <omp.h>
….
#pragma omp parallel for num_threads(4)
  for(i=0;i<m;i++) {
    código C correcto (no código omp)
    #pragma omp parallel for num_threads(4)
      for(j=0;j<n;j++)
        código C correcto (no código omp)
….
```

Respuesta
Si el paralelismo anidado no está activado 4 hilos, 16 en caso contrario.

Pregunta. En el código adjunto de la pregunta anterior, si el número de cores es 4, ¿es más eficiente que el correspondiente eliminando el segundo #pragma? Razonar la respuesta.

Respuesta
No, independientemente de que el paralelismo anidado esté o no habilitado, dado que implicaría mayores costes de creación/destrucción, conmutación, etc. de hilos, sin la contrapartida de disponer de un número de *cores* suficiente para no incurrir en compartición de recursos de computación.

Pregunta. ¿Es correcto el siguiente fragmento de código?

```
NC=omp_get_num_threads()
#pragma omp parallel
  #pragma omp for num_threads(NC)
...
```

Respuesta
SÍ

Pregunta. ¿Qué valor imprime si el hilo de identificador 0 se ejecuta primero? Razone la respuesta.

```
#include <omp.h>
int main() {
  int x=2;
  #pragma omp parallel num_threads(2) shared(x)
      if (omp_get_thread_num() == 0)
        x = 5;
      else
        printf("%d\n", x);
  return 0;
}
```

Respuesta
Imprime 5. Nota: podría imprimir 2 bajo ciertas condiciones (hardware+software) debido al modelo de memoria de OpenMP (coherencia relajada), ver #pragma omp flush(x)

Pregunta. ¿Qué valor se imprime? Razone la respuesta.

```
#include <omp.h>
int main() {
  int x=0;
  #pragma omp parallel num_threads(1)
    #pragma omp sections
    {
      #pragma omp section
        { x++;       printf("%d\n", x); }
      #pragma omp section
        { x++;       printf("%d\n", x); }
    }
  return 0;
}
```

Respuesta
Solo hay un hilo que ejecuta ambas secciones, por lo que se imprime 1 y 2.

Pregunta. Imprime "1 2" y se quiere "1 1". Solucionarlo sin modificar el código nativo *C*.

```
#include <omp.h>
int main() {
  int x=0;
  #pragma omp sections firstprivate(x)
  {
    #pragma omp section
      { x++;  printf("%d ", x);  }
    #pragma omp section
      { x++;  printf("%d ", x);  }
  }
  return 0;
}
```

Solución

```
#include <omp.h>
int main() {
  int x=0;
    #pragma omp parallel sections firtsprivate(x)
    {
      #pragma omp section
      { x++;  printf("%d ", x);  }
      #pragma omp section
      { x++;  printf("%d ", x);  }
    }
  return 0;
```

Pregunta. Este código. ¿Qué imprime? Razone la respuesta.

```
NC=omp_get_num_threads();
#pragma omp parallel for num_threads(NC)
  for(i=0;i<10;i++)
    printf("%d\n", omp_get_thread_num());
…
```

Respuesta
Imprime el valor 0 diez veces: solo hay un hilo en ejecución.

Pregunta. Este código. ¿Qué imprime? Razone la respuesta en el espacio habilitado.

NC=omp_get_num_threads();
#pragma omp parallel for
  for(i=0;i<10;i++)
  printf("%d\n", omp_get_thread_num());
…
	Respuesta
	10 veces valores entre 0 y:
	a) el número de *cores* menos 1 o
	b) el número de procesos establecidos con las variables de entorno y/o por el sistema

Pregunta. Si antes de ejecutar el siguiente código en el *shell* se exporta ***OMP_NUM_THREADS=4*** ¿Cuántos hilos serán desplegados en la ejecución? ¿Por qué?

omp_set_num_threads(2); #pragma omp parallel for num_threads(3) …	**Respuesta** 3, dado que la cláusula es de menor alcance y mayor prioridad

Pregunta. ¿Cuánto más rápido es, potencialmente, el código adjunto que el secuencial? Razonar la respuesta.

…
#pragma omp for num_threads(4)
  for(i=0;i<n;i++)
  "lo que se quiera hacer"
…
	Respuesta
	Es más lento que el secuencial, o igual de rápido en el mejor de los casos. Nótese que no hay directiva "parallel", por lo que no hay ejecución paralela.

Pregunta. ¿Es correcto el siguiente fragmento de código?

int i, j=0, Carga;
/* inicializar adecuadamente M y Cores */
Carga=M/Cores;
#pragma omp parallel
  #pragma omp single nowait
  for (i=0; i<Cores; i++)
  {
  …
  #pragma omp task untied
  {
  for (j=0; j<Carga; j++)
  …
	Respuesta
	No, la variable *j* debería privada.

Pregunta. ¿Es correcto el siguiente fragmento de código?

#pragma omp parallel private(i)
  #pragma omp sections
  {
  #pragma omp section
  for (i=0; i<M/3; i++)
  ...
  #pragma omp section
  for (i=M/3; i<2*M/3; i++)
  ...
  #pragma omp section
  for (i=2*M/3; i<M; i++)
  #pragma omp task untied
  ....
  }
	Respuesta
	Sí

Pregunta. ¿Es correcto el siguiente fragmento de código?

```
#pragma omp parallel
  #pragma omp sections
  {
    for(i=0; i<omp_get_num_threads(); i++)
      #pragma omp section
          ...
```

Respuesta
No. Sintácticamente es incorrecto

Pregunta. ¿Es correcto el siguiente código? SÍ/NO ¿Por qué? ¿Qué valor imprime?

```
#include <omp.h>
int main() {
  int x=0;
  #pragma omp parallel sections
  {
    #pragma omp section
      #pragma omp critical
        #pragma omp parallel
          #pragma omp single
                    x++;
  }
  printf("%d\n", x);
  return 0;
}
```

Solución
Crea una región paralela con una sola sección luego un solo hilo en ejecución. Ese hilo ejecuta *critical* y encuentra otro *parallel*. Que esté, o no, el paralelismo anidado activado es irrelevante puesto que la sentencia *x++* está dentro de *single*, donde solo entra uno.

Sí, es correcto. Imprime 1.

Pregunta. De los siguientes fragmentos de código ¿Cuál/cuáles es/son correcto/correctos? Razone la respuesta en el espacio habilitado.

```
int i, M=8;
#pragma omp parallel private(i)
  #pragma omp sections
  {
    #pragma omp section
      for(i=0; i<M/2; i++)
        …
    #pragma omp section
      for(i=M/2; i<M; i++)
        #pragma omp task untied
          …
  }
```

```
int i, M=8;
#pragma omp parallel private(i)
  #pragma omp sections
  {
    #pragma omp section
      for(i=0; i<M/2; i++)
        …
    #pragma omp section
      #pragma omp single nowait
        for(i=M/2; i<M; i++)
          …
  }
```

Respuesta
El de la izquierda es correcto. El otro no por el uso de "single". Un *section* no puede tener anidado un single (*section* no puede tener otro de constructor de carga). Si en medio hay otro *pragma* (por ejemplo *parallel*) ya es otro contexto, región, por lo que sí se podría.

Pregunta. Considere el código de la figura. ¿Dónde colocaría "*#pragma omp parallel for*"?

```
for (i=1; i<N; i++)
  for (j=0; j<N; j++)
    A[i, j]=MIN(A[i-1, j], A[i-1, j-V[i]]);
```

Respuesta
En el bucle *i* sería errónea porque los valores de *A[i, j]* dependen de los de *A[i-1, j]*. Respecto al bucle en *j* ver respuesta a la siguiente pregunta.

Pregunta. Considere el código de la figura. ¿Dónde colocaría "*#pragma omp parallel for*"?

```
for (i=1; i<N; i++)
  for (j=0; j<N; j++)
    A[i, j]=MIN(A[i, j], A[i, j-V[i]]);
```

Respuesta
En el bucle *i*. Respecto al bucle *j* la pregunta es ambigua: se desconoce la semántica de "*j-V[i]*". En el supuesto de que sí se pudiese sería menos

	eficiente por la escasa intensidad computacional de la sentencia.

Pregunta. Considere el siguiente código donde el tiempo de ejecución de *f(i)* es proporcional al valor de *i*. Obtenga con "*#pragma omp parallel for*" una solución eficiente.

for (i=0; i<N; i++) f(i);	**Respuesta** #pragma omp parallel for schedule(guided) o "dynamic" serían apropiados

Pregunta. Considere el siguiente código donde el tiempo de ejecución de *f(i)* es inversamente proporcional al valor de *i*. ¿Es posible con "*#pragma omp parallel for*" una solución eficiente?

for (i=0; i<N; i++) f(i);	**Respuesta** Sí, cambiando la forma de recorrer la i en el for: *for (i=N-1; i>=0; i--)*

Pregunta. ¿Es correcto, aunque sea ineficiente? Razonar la respuesta.

…
#pragma omp parallel
  #pragma omp task untied
  comp_fib_numbers(n-1)
…
	Respuesta
	No, no lo es. Falta la directiva "#pragma omp single", o equivalente, entre el "#pragma omp parallel" y el "#pragma omp task untied". Tal como está codifica se resuelve p veces el problema, en vez de una, siendo p el número de cores activos del sistema.

Pregunta. Considere el código mostrado en la figura. ¿Podría un compilador optimizar (paralelizar y/o vectorizar) sin dificultad? ¿Por qué?

for (i=1; i<N; i++) {
  X[i] = Y[i-1] * 2;
  Y[i] = Y[i] + i;
}
	Respuesta
	Probablemente No al detectar Dependencia de Flujo (la salida de la iteración *i* es entrada de la iteración *i+1*.

Pregunta. Considere el código de la figura. Paralelizarlo (*Y* ha sido inicializado previamente)

for (i=1; i<N; i++)
{
  X[i] = Y[i-1] * 2;
  Y[i] = Y[i] + i;
}
	Respuesta
	#pragma omp parallel for
	for (i=1; i<N; i++)
	  Y[i]=Y[i] + i;
	#pragma omp parallel for
	for (i=1; i<N; i++)
	  X[i]=Y[i-1] * 2;

Pregunta. Sea un sistema con "coherencia de caché", 2 *cores* y capacidad para ejecutar 2 hilos a la vez. ¿El algoritmo adjunto presentar algún tipo de pérdida de rendimiento? SÍ/NO ¿Por qué?

En paralelo P0 y P1 hacen
  k = obtener_mi_id(); /* k=0 para P0 y k=1 para P1 */
  S[k] = 0.0;
  for (i=0; i<n; i+=2) /* n es par */
  si (k==0)
  S[k] += b[i];
  else
  S[k] += b[i+1];
	Respuesta
	Sí, si se produce *False Sharing*: el mismo bloque de caché con los datos necesarios para los dos hilos concurrentes y, para mantener la coherencia, se invalida todo el bloque.

Pregunta. Sea un sistema con "coherencia de caché" y 2 *cores*. El algoritmo de la izquierda puede presentar *False Sharing* ¿Y el de la derecha? SÍ/NO ¿Por qué?

En paralelo P0 y P1 hacen
  k = obtener_id(); /* k=0 en P0 y k=1 en P1 */
  S[k] = 0.0;
  for (i=0; i<n; i+=2) /* n es par */
  si (k==0)
  S[k] += b[i];
  else
  S[k] += b[i+1];
	En paralelo P0 y P1 hacen
	  k = obtener_id(); /* k=0 en P0 y k=1 en P1 */
	  S[k] = 0.0;
	  for (i=0; i<**n/2**; i+=2) /* n es par */
	  si (k==0)
	  S[k] += b[i];
	  else
	  S[k] += b[**n/2+i**];

Respuesta Sí si el tamaño del bloque de cache es mayor que **n/2**; no en el otro caso.	

(Curso 2019 – 2020)

Pregunta. Sea el código adjunto. En total, ¿Cuántos hilos se despliegan? Razone la respuesta.

#pragma omp parallel for num_threads(3)
for(int i=0; i<3; i++)
  printf("%d\n", i);
  #pragma omp parallel for num_threads(5)
  for(int j=0; j<3; j++)
  printf("%d\n", j);
	Respuesta
	En total se despliegan 8, primero 3 y luego 5, no hay anidamiento. Nótese que 2 de ellos, en el segundo *pragma*, no tienen tareas que realizar (se pueden activar o no)

Pregunta. El código adjunto ¿calcula correctamente *numbers(3)*? ¿qué imprime?

int numbers(int n) {
  int a;
  if (n < 0) return 0;
  #pragma omp task shared(a)
  a = n + numbers(n-1);
  #pragma omp taskwait
  return a;
}
…
#pragma omp parallel
  printf("%d\n", numbers(3));
…
	Respuesta
	Sí. Imprime 6, tantas veces como hilos se despliegen en la directiva "#pragma omp parallel"
	
	Complementario (no necesario incluir en la respuesta). El número de veces que se imprime 6 depende de factores externos como el número de cores, el valor de las variables de entorno relacionadas, si previamente se ha llamado a funciones relacionadas de la API, etc.
	No es eficiente, por todo lo dicho. Seguramente lo que se pretendía era incluir, antes de "printf("%d\n", numbers(3));", "#pragma omp single nowait" o "#pragma omp master".

(Curso 2020 – 2021)

Pregunta. Sean los siguientes programas. Completar la frase de la última fila de la tabla (modelos A y B).

Fragmento Código Izquierda	Fragmento Código Derecha

"declaraciones y definiciones necesarias"
for (i=0; i<n; i++)
  V[i] = A[i]*B[i];
	"declaraciones y definiciones necesarias"
	for (i=0; i<n; i+=32)
	  V[i] = A[i]*B[i];

Respuesta El Fragmento Código Izquierda es apto para CPU, porque explota la **Localidad Espacial**, mientras que el Fragmento Código Derecha lo es para GPU porque explota la **Coalescencia**.	

Pregunta. Escriba en "PARA CPU" la directiva de OpenMP más sencilla para paralelizar el código de la pregunta anterior para CPU. En "PARA GPU" complete la parte que falta en "<<<conocido, ¿ ?>>>" de una invocación correcta a un kernel que podría corresponder con el código de la pregunta anterior apto para GPU (modelos A y B).

PARA CPU	PARA GPU
#pragma omp parallel for	<<<conocido, **32**>>>

Pregunta. Complete la siguiente frase (modelos A y B).

El ámbito de las estrategias de asignación dinámicas basadas en algoritmos de planificación es la **Descomposición funcional**.

Pregunta. Sea el siguiente fragmento de código Si se quiere que se imprima por pantalla cada número entero entre 0 y n, sin importar el orden, ¿Es correcto? Razone la respuesta (modelo A).

...
#pragma omp parallel
  #pragma omp single
  {
  int i;
  for(i=0;i≤n;i++)
  #pragma omp task
  printf("el valor de i es %d\n", i);
  }
…
	Respuesta
	Si. Solo uno de los hilos entrará en el "#pragma omp single" y lanzará $n+1$ tareas, con un valor entre 0 y n para la variable i. Cada tarea será ejecutada por un hilo.

Pregunta. Sea el siguiente fragmento de código Si se quiere que se imprima por pantalla cada número entero entre 0 y n, sin importar el orden, ¿Es correcto? Razone la respuesta (modelo B).

...
#pragma omp parallel
  {
  int i;
  for(i=0;i≤n;i++)
  #pragma omp task
  printf("el valor de i es %d\n", i);
  }
…
	Respuesta
	No. TODOS los hilos ejecutan "#pragma omp task" y cada uno de ellos lanzará $n+1$ tareas, con un valor entre 0 y n para la variable i. Por tanto, cada tarea (valor de i) será ejecutada tantas veces como hilo. Se imprime p veces cada valor de i, siendo p el número de hilos.

CEX2: Preguntas relativas a CUDA

Pregunta. ¿Es correcto? ¿Mejor o peor que la otra opción? Razonar la respuesta.

```
__global__ void kernel(…) {
  __shared__ float sData[256];
  …
}
void main() {
  ...
  kernel<<<nB, nT, nT*4>>>(…);
}
```

Respuesta
No es correcto. Es sintácticamente incorrecto. O sobra el "nT*4" al lanzar el kernel o sobra el "256" y falta extern en el kernel. Ni mejor ni peor, está mal. Si estuviese bien y se pudiese hacer, por la naturaleza del problema, es mejor la declaración estática (tiempo de compilación).

Pregunta. ¿Es correcto? Razonar la respuesta.

```
__global__ void kernel(…) {
  extern __shared__ float smem[];
  …
}
void main() {
  ...
  kernel<<<nB, nT>>>(…);
}
```

Respuesta
No, es sintácticamente incorrecto. Para lanzar el kernel se debería usar "kernel<<<nB, nT, 3er argumento>>>" dado que se trata de una declaración dinámica para la memoria compartida. "3er argumento" es el tamaño para "smem".

Pregunta. Sea una GPU con las siguientes características técnicas "*Max dimension size of a thread block (x,y,z): (64, 64, 1). Max dimension size of a grid size (x,y,z): (64, 64, 64)*". Sea un vector de tamaño n que hay que normalizar. Si n es 2^{18} indique la estrategia de hilos y bloques más razonable.

Respuesta
Por la naturaleza de los datos, un vector, y el tamaño podría ser bloques de (64, 1, 1) y grid de (64, 64, 1). Con ello se tiene $2^6 * 2^6 * 2^6 = 2^{18}$ (64 es 2^6)

Pregunta. Sea una GPU como las de prácticas y un vector V. Sea un *kernel* que se ejecuta como: $kernel <<< BG, 2048 >> (V, \dots)$. ¿*cuda-memcheck* reportaría algún error? Razone la respuesta.

Respuesta
Sí, algo similar a "… parámetro inválido", consecuencia de que el número de hilos por bloque supera el máximo admitido por bloque

Pregunta. Supóngase que el código de la pregunta anterior es correcto. Se quiere hacer $y[i] = \sum_0^{n-1} x[i]\,, \forall\, 0 \le i < n$. Sea el kernel adjunto. Escribir la modificación mínima en el código de la pregunta anterior para que el kernel resuelva correctamente el problema deseado.

```
__global__ void sum(int *x, int *y) {
  __shared__ int copia[4];
  unsigned int
id=blockIdx.x*blockDim.x+threadIdx.x;

  copia[id]=x[id];
  __syncthreads();

  for (unsigned int i=0; i<4; i++) y[id] += copia[i];
}
```

Respuesta
…<<<**1, 4**>>>…
Sin la modificación los elementos de y valdrán la mitad del valor deseado. La memoria *shared* no se comparte entre hilos de distinto bloque.

Pregunta. El siguiente fragmento de código ¿qué valor imprime, aproximadamente, sabiendo que la duración del *kernel* es 120 segundos?

Ctimer(&elapsed, &ucpu, &scpu, 0); kernel<<<Grid, Block>>>(param1, …, paramn); Ctimer(&elapsed, &ucpu, &scpu, 1); printf("\nTiempo %f segundos\n", elapsed); …	**Respuesta** Un valor próximo a 0. La ejecución de los *kernels* en GPU es asíncrona respecto a la CPU (la llamada al *kernel* no bloquea la ejecución del código CPU).

Pregunta. Sea el siguiente programa. ¿Es correcto? ¿Por qué?

int main() {
  int *x, *y, i;
  cudaMallocManaged(&x, 4*sizeof(int));
  cudaMallocManaged(&y, 4*sizeof(int));
  x[0]=x[1]=x[2]=x[3]=1;
 	
  sum<<<2, 2>>>(x, y);
 	
  for (i=0; i<4; i++) printf("%d\n", y[i]);
  return 0;
}
	Respuesta
	No, por varias causas. Una de ellas es "violación de segmento". La ejecución del *kernel* es asíncrona y antes de que el *kernel* termine se ejecuta el bucle *for*. Si no diese *core* el contenido de *y* sería inestable.
	
	Falta *cudaDeviceSynchronize*() después de la llamada al *kernel*.

(Curso 2019 – 2020)

Pregunta. Sea la siguiente llamada a un *kernel* CUDA: "kernel<<<<***A***, ***B***, ***C***, ***D***>>>(…);" ¿Qué denotan los parámetros ***C*** y ***D***. ¿Cuándo se usan?

Respuesta ***C*** es el tamaño, en bytes, de la memoria *shared* cuando se usa y el tamaño se determina en tiempo de ejecución. Si no se usa se puede omitir, o poner 0 en caso de que se use ***D***. ***D*** es el *stream* sobre el que se lanzará el *kernel*. Si se omite se asigna al *stream* por defecto, el 0.

Pregunta. Sea el siguiente fragmento de código, donde M es 0xffffffff. En las fases tempranas de "Fundamentos del Diseño de Algoritmos Paralelos" ¿Qué estrategia es equivalente? Citar nombre, objetivo y cuando se usa.

__inline__ __device__ int R(int v) {
  for (int f=16; f>0; f/=2)
  v +=__shfl_down_sync(M, v, f);
  return v;
}
	Respuesta
	Se usa para eliminar comunicaciones centralizadas. El objetivo es maximizar la concurrencia. La estrategia, general, se llama "divide y vencerás".

(Curso 2020 – 2021)

Fragmento Código Izquierda	Fragmento Código Derecha

"declaraciones y definiciones necesarias"
for (i=0; i<n; i++)
  V[i] = A[i]*B[i];
	"declaraciones y definiciones necesarias"
	for (i=0; i<n; i+=32)
	  V[i] = A[i]*B[i];

Respuesta El Fragmento Código Izquierda es apto para CPU, porque explota la **Localidad Espacial**, mientras que el Fragmento Código Derecha lo es para GPU porque explota la **Coalescencia**.	

Pregunta. Escriba en “PARA CPU” la directiva de OpenMP más sencilla para paralelizar el código de la pregunta anterior para CPU. En “PARA GPU” complete la parte que falta en “<<<conocido, ¿ ?>>>” de una invocación correcta a un kernel que podría corresponder con el código de la pregunta anterior apto para GPU (modelos A y B).

PARA CPU	PARA GPU
#pragma omp parallel for	<<<conocido, **32**>>>

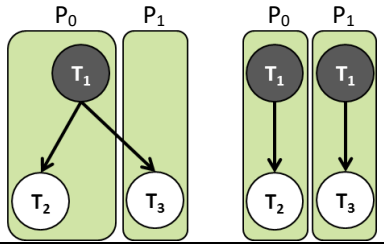
Pregunta. Complete la siguiente frase (modelos A y B).

La coalescencia es un concepto relacionado con los hilos del **warp**, mientras que para la memoria compartida lo relevante son los hilos del **bloque**.

CEX2: Preguntas de Teoría

Pregunta. Citar una ventaja de replicar la computación. Ponga un ejemplo gráfico

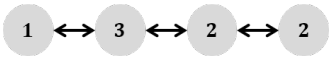
Respuesta
Eliminar dependencias o comunicaciones. Por ejemplo, pasar de la figura de la izquierda a la de la derecha.



Pregunta. Sea un vector v de dimensión n sobre el cual se ha aplicado un particionado del dominio resultando que todas las comunicaciones son entre vecinos. Al aplicar la fase de agrupamiento se sigue la misma estrategia que en los ejemplos realizados en clase. ¿Qué nombre recibe la estrategia que procede de igual forma en la fase de asignación?

Respuesta
Bisección Recursiva Coordinada. En cada paso (corte), divide el grafo en 2 partes iguales (punto medio).

Pregunta. Sea el grafo de tareas de la figura. Se desea resolver la asignación en un sistema con 2 procesadores. ¿Qué sería mejor, aplicar "Bisección Recursiva Coordinada" o "Bisección Recursiva no Balanceada"? ¿Por qué?



Respuesta
Indistintamente. "Bisección Recursiva Coordinada" divide por la mitad y "Bisección Recursiva no Balanceada" busca la división (1/3, 2/2, 3/1) de menor coste, que corresponde con la mitad.

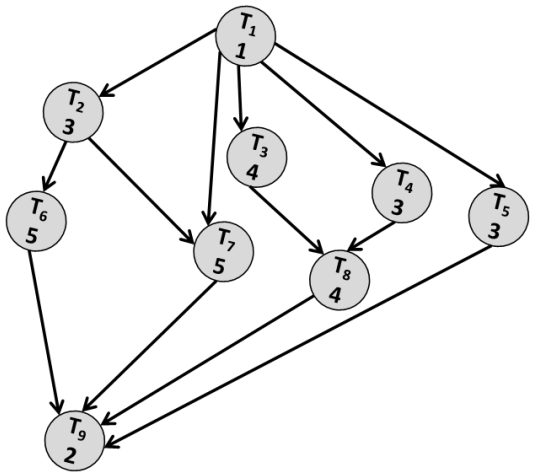
Pregunta. La granularidad se puede cuantificar como $G(n,p) = (t\_cal\ (n,p))\ /\ (t\_com\ (n,p))$. ¿Con qué otro nombre se conoce a su inversa en el diseño de los algoritmos para memoria distribuida?

Respuesta
Relación Superficie-Volumen

Pregunta. La relación "Superficie Volumen" que se obtiene en uno de los posibles agrupamientos para un diseño dado es ***0*** ¿Es factible? Razone la respuesta.

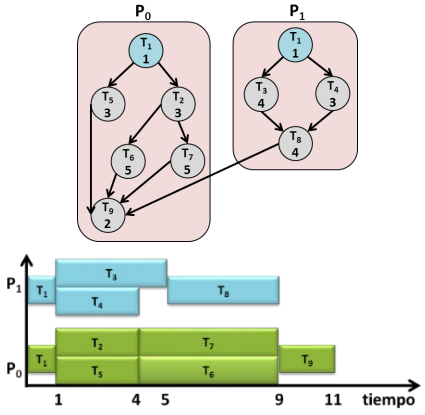
Respuesta
Sí, cuando al agrupar se eliminan todas las comunicaciones el numerador de la relación "Superficie Volumen" es 0. Es un síntoma de un agrupamiento óptimo

Pregunta. Sea el grafo de dependencias adjunto. Suponiendo que se dispone de 2 procesadores *dual-core* agrupe las tareas de forma que se minimice el tiempo de ejecución. Esboce el cronograma temporal después de agrupar.

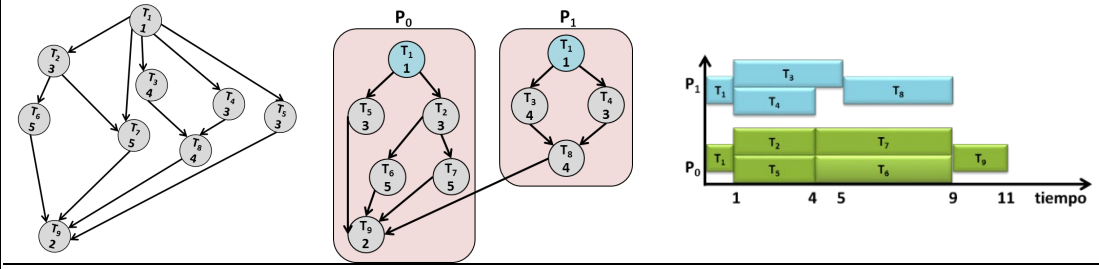


Respuesta

Replicar la tarea 1 en todos los procesadores para eliminar la dependencia. Lo siguiente es la distribución de las tareas tratando de buscar el mayor grado de concurrencia con el menor número de comunicaciones. El resultado se muestra en la figura adjunta.



Pregunta. En algún examen se presentaba el grafo de dependencias adjunto (figura de la izquierda) y se pedía la agrupación de tareas (figura centro) que minimiza el tiempo de ejecución y su cronograma temporal (figura derecha), para un sistema con 2 procesadores dual-core. ¿Considera que las unidades del eje X de la respuesta dada (cronograma temporal) son correctas si fuese un sistema formado por 2 nodos cada uno de ellos con un único procesador dual-core? Razonar la respuesta.



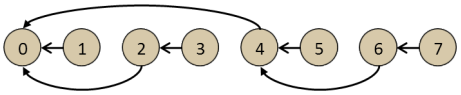
Respuesta

NO. El tiempo final sería mayor puesto que habría que añadir el tiempo de la comunicación necesaria para transferir desde P_1 a P_0 (o desde P_0 a P_1) el resultado parcial obtenido P_1 (o P_0).

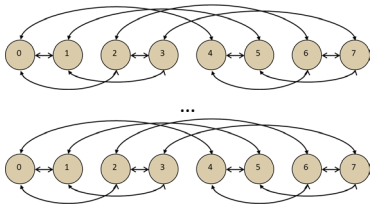
Pregunta. Sean A y B matrices cuadradas de dimensiones nxn y el código secuencial de la figura adjunta. Sea el grafo de dependencias mostrado en la figura el correspondiente a una fila genérica, cuando $n = 8$, y hay tantos procesadores como elementos en las matrices. Construya el grafo de dependencias completo

para un valor genérico de n subsanando las deficiencias observadas en el grafo adjuntado. Proponga un agrupamiento óptimo cuando el número de procesadores es mucho menor que n.

```
for (i=0; i<n; i++)
{
  tmp = 0.0;
  for (j=0; j<n; j++)
    tmp += B[i, j] * A[i, j];
  for (j=0; j<n; j++)
    B[i, j] = tmp;
}
```



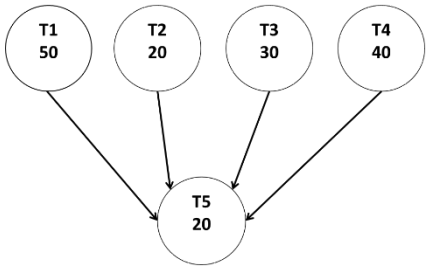
Respuesta



Agrupamiento por bloques de filas consecutivas eliminando, de esta forma, las comunicaciones.

(Curso 2020 – 2021)

Pregunta. Sea el diagrama adjunto. Calcular $C(n,4)$, $T_0(n,4)$, $S(n,4)$ y $E(n,4)$.



Respuesta

$T(n) = 50 + 20 + 30 + 40 + 20 = 160$

$T(n,p) = max(50,20,30,40) + 20 = 50 + 20 = 70$

$C(n,4) = 4T(n,p) = 4x70 = 280$

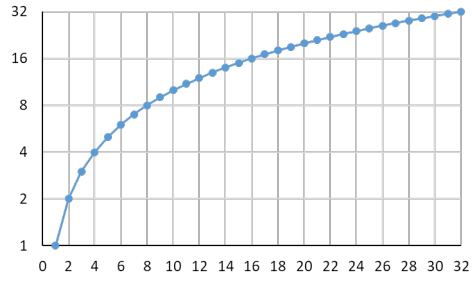
$T_0(n,4) = C(n,4) - T(n) = 280 - 160 = 120$

$S(n,4) = T(n)/T(n,4) = 160/70 = 16/7 = 2.29$

$E(n,4) = S(n,4)/4 = 2.29/4 = 0.57$

(Igual que *slide* 21 de los apuntes)

Pregunta. En la gráfica, donde el eje X es p y el eje Y es el $S(n,p)$, ¿Cuál es el valor de $\alpha(n)$ de la ley de *Amdahl*? Razonar la respuesta.



Respuesta **(Directamente de los apuntes)**

$$S(n,p) = \frac{T(n)}{T(n,p)} = \frac{p}{(p-1)\alpha(n)+1}$$

Como en la gráfica el $speedup = p$ esto solo puede ser si $\alpha(n) = 0$

(Directamente de los apuntes, *slide* 26 de los apuntes)

Problemas

Problema 1

Sea un dominio definido por una matriz de dimensiones ZxN con $Z \leq N$, como se muestra en Figura 1. Sea p el número de procesadores que se pueden configurar en línea o como una malla bidimensional no cerrada. p tiene raíz exacta y las dimensiones son divisibles tanto por p como por $\sqrt{p}$.

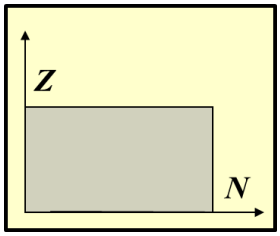


Figura 1

Figura 2

El problema consiste en iterar k veces actualizando todos y cada uno de los puntos del dominio en cada iteración. Al actualizar cada punto, en la iteración $i+1$, intervienen los valores de sus vecinos en la iteración i según muestra la Figura 2. Actualizar 1 punto = 1 flop.

El problema será resuelto en un sistema paralelo. Son varios los particionados 1D / 2D posibles. El alumno debe seleccionar el mejor particionado justificando su elección mediante la realización de un estudio teórico completo. Realizar el análisis de la escalabilidad.

Problema 2

Igual a Problema 1. con comunicaciones sobre el eje X, como se muestra en la figura adjunta.

Problema 3

Igual a Problema 1. Con un dominio definido por un volumen de ZNN puntos, con $Z \leq N$ (ver Figura 3). Las comunicaciones/dependencias son las que se muestran en la Figura 3. El coste de actualizar un punto un *flop*. El problema debe ser resuelto en un sistema paralelo de memoria distribuida siendo p el número de procesadores o nodos del sistema.

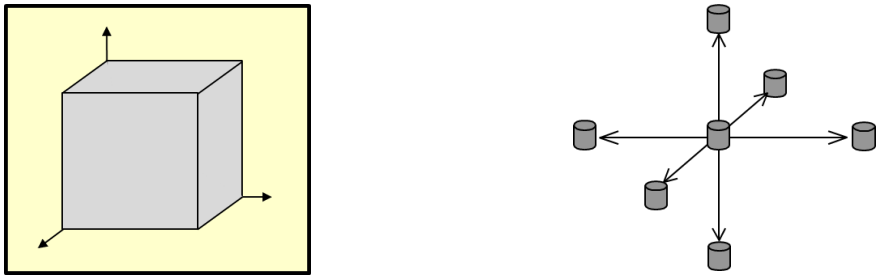


Figura 3. Izquierda Dominio. Derecha esquema de vecindad/dependencias

Problema 4

Igual a Problema 3. Pero los p procesadores/nodos están dispuestos como una malla bidimensional no cerrada tal que p tiene raíz exacta y N es divisible por $\sqrt{p}$. Seleccionar el mejor particionado 2D justificando su elección con un estudio teórico completo.

Solución Problema 1

A la hora de realizar el estudio se puede suponer, por simplicidad y sin pérdida de generalidad, que el número de iteraciones es uno ($k = 1$).

Como se ha visto en clase, la mejor opción es realizad un particionado del dominio. Considerando las dependencias/comunicaciones indicadas en el enunciado el grafo resultante es el que se muestra en Figura 4. Todas las tareas realizan, simultáneamente, la misma computación y mismas comunicaciones.

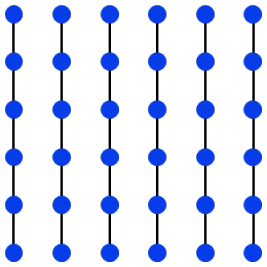


Figura 4

Para realizar el agrupamiento, considerando la regularidad del dominio (matriz) y lo dicho en el párrafo anterior, los particionados factibles son: dividir el dominio según el eje N, según el eje Z o usando ambos ejes.

De los particionados 1D (Figura 5 y Figura 6) el primero (Figura 5) es tal que todos los elementos necesarios para realizar los cálculos están en el mismo contexto: no hay dependencias/comunicaciones. En ambos particionados 1D el número de elementos por procesador es ZN/p.

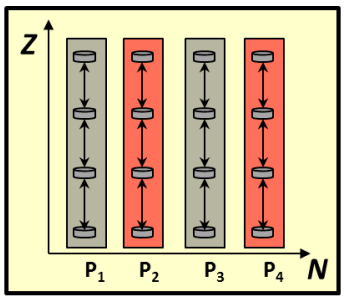


Figura 5

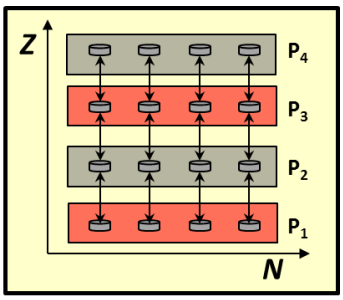


Figura 6

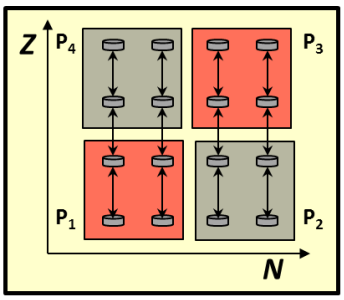


Figura 7

Respecto al particionado 2D (Figura 7) en cada procesador habría exactamente $\frac{Z}{\sqrt{p}}\frac{N}{\sqrt{p}} = \frac{ZN}{p}$ puntos que coincide con los particionados 1D.

Por la relación superficie volumen se sabe que:

- Particionado 1D sobre el eje $N \Rightarrow \frac{p*0}{ZNt_c} = 0$
- Particionado 1D sobre el eje $Z \Rightarrow \frac{2p(t_s+Nt_w)}{ZNt_c}$
- Particionado 2D sobre el eje $Z \Rightarrow \frac{2p\left(t_s+\frac{N}{\sqrt{p}}t_w\right)}{ZNt_c}$

Consecuentemente el mejor particionado es 1D sobre el eje N, y el peor el 1D sobre el eje Z. El estudio teórico completo del particionado seleccionado es:

$$T(n) = T(n,1) = ZNt_c$$

Nótese que, al no haber dependencias, el reparto ser balanceado y trabajar todos los procesadores concurrentemente no hay esperas ni sobrecargas. Por todo ello el tiempo paralelo es:

$$T(n,p) = \frac{ZN}{p}t_c$$

Consecuentemente,

$$E(n,p) = \frac{ZNt_c}{ZNt_c} = 1$$

y la función de sobrecarga nula (su coste coincide con el tiempo de ejecución secuencial).

Para el particionado 1D sobre el eje Z

$$T(n,p) = \frac{ZN}{p}t_c + 2(t_s + Nt_w)$$

$$E(n,p) = \frac{ZNt_c}{ZNt_c + 2p(t_s + Nt_w)} = \frac{ZNt_c}{ZNt_c + 2pt_s + 2pNt_w}$$

Y para el particionado 2D

$$T(n,p) = \frac{ZN}{p}t_c + 2\left(t_s + \frac{N}{\sqrt{p}}t_w\right)$$

$$E(n,p) = \frac{ZNt_c}{ZNt_c + 2p\left(t_s + \frac{N}{\sqrt{p}}t_w\right)} = \frac{ZNt_c}{ZNt_c + 2pt_s + 2Nt_w\sqrt{p}}$$

Solución Problema 2

El análisis sería similar, y el resultado el mismo, usando como particionado el mostrado en la Figura 6.

Solución Problema 3

Como se ha visto en la resolución de los ejercicios anteriores, una consecuencia importante de cualquier particionado sobre uno de los ejes es que elimina las dependencias/comunicaciones de los otros dos ejes y que, además, las que persisten son locales (entre vecinos). Visualmente en Figura 8 se recoge el resultado del particionado realizado sobre uno de los ejes en N.

Otra consecuencia es que al ser dos dimensiones iguales y una menor (Z), la menor no se debe usar para el particionado porque amplificaría el impacto de las dependencias/comunicaciones.

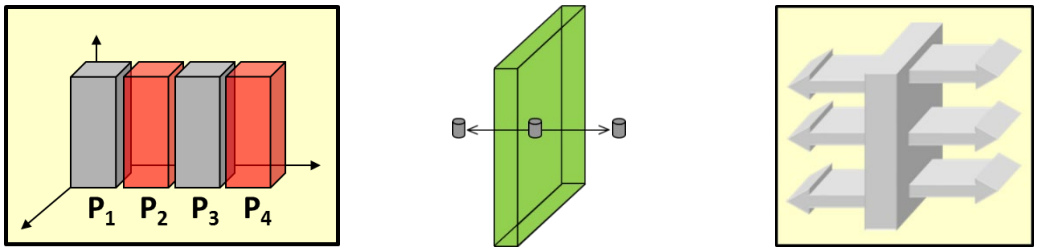


Figura 8. Izquierda ejemplo de agrupamiento sobre uno de los ejes en N con p=4. Centro comunicaciones a nivel de elemento y derecha a nivel de procesador.

Entonces:

- Tiempo de ejecución secuencial: $T(n) = ZN^2t_c$
- Tiempo de ejecución paralelo[1]: $T(n,p) = \frac{ZN^2}{p}t_c + 4(t_s + ZNt_w) - 0 + 0$
- Speedup: $S(n,p) = \frac{pN^2Zt_c}{N^2Zt_c + 4p(t_s + ZNt_w)}$
- Eficiencia: $E(n,p) = \frac{N^2Zt_c}{N^2Zt_c + 4p(t_s + ZNt_w)}$
- Coste: $C(n,p) = N^2Zt_c + 4p(t_s + ZNt_w)$
- Función de sobrecargar (*overhead*): $T_0(n,p) = 4pt_s + 4pZNt_w$

Respecto al análisis de la escalabilidad ver diapositiva 42 del tema CEX correspondiente.

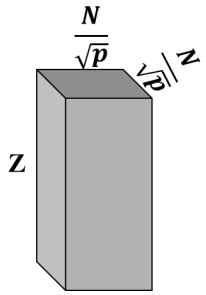
Si se hubiese utilizado el eje en Z para el particionado la función de sobrecarga sería $T_0(n,p) = 4pt_s + 4pNNt_w$, donde se observa que peso de las comunicaciones es mayor.

Solución Problema 4

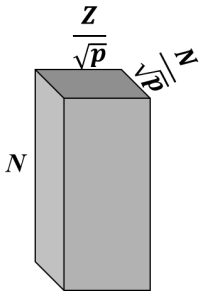
La solución es similar a la de los problemas anteriores, con la salvedad de las comunicaciones. Al ser un particionado 2D los procesadores intermedios tienen 4 superficies de contacto que según que 2 ejes se usen para la división del problema resulta que:

[1] El número 4 introducido en la expresión del tiempo de ejecución paralelo proviene de considerar que el intercambio (enviar / recibir) entre 2 procesadores son 2 operaciones de comunicación. Si el intercambio se contabiliza como una única operación habría que sustituir el multiplicador 4 por 2 en todas las expresiones.

Caso 1



Caso 2



Para el Caso 1 $T(n,p) = \frac{ZN^2}{p}t_c + 4\left(t_s + \frac{ZN}{\sqrt{p}}t_w\right)$. Para el Caso 2 $T(n,p) = \frac{ZN^2}{p}t_c + 2\left(t_s + \frac{ZN}{\sqrt{p}}t_w\right) + 2\left(t_s + \frac{N^2}{\sqrt{p}}t_w\right)$. Como $N \geq Z$ sus comunicaciones pesan más y, por tanto, la elección debe ser el Caso 1. Entonces para el Caso 1:

- Tiempo de ejecución secuencial: $T(n) = ZN^2t_c$
- Tiempo de ejecución paralelo: $T(n,p) = \frac{ZN^2}{p}t_c + 4\left(t_s + \frac{ZN}{\sqrt{p}}t_w\right)$
- Speedup: $S(n,p) = \frac{pZN^2t_c}{ZN^2t_c + 4\sqrt{p}(\sqrt{p}t_s + ZNt_w)}$
- Eficiencia: $E(n,p) = \frac{ZN^2t_c}{ZN^2t_c + 4\sqrt{p}(\sqrt{p}t_s + ZNt_w)}$
- Coste: $C(n,p) = ZN^2t_c + 4\sqrt{p}\left(\sqrt{p}t_s + ZNt_w\right)$
- Función de sobrecargar (*overhead*): $T_0(n,p) = 4\sqrt{p}\left(\sqrt{p}t_s + ZNt_w\right)$

Problema 5

Una empresa tenía que resolver un problema que, formalmente, consistía en un sistema de ecuaciones $Ax = b$ donde $A \in \mathcal{R}^{nxn}, x \in \mathcal{R}^n, b \in \mathcal{R}^n; n > 1$. Un especialista diseñó un algoritmo secuencial. Recientemente la empresa ha comprado un sistema de computación de memoria distribuida y quiere paralelizarlo. La única documentación que existe de la versión secuencial es el código que se muestra a continuación.

```
for (i = 0; i < n-1; i++)                  /* De primera columna a la penúltima      */
    for (j = n-1; j > i; j--)              /* De última fila a columna actual + 1    */
    {
        for (k = i+1; k < n; k++)          /* De columna actual + 1 a final (n - 1)  */
            A[j,k]=funcion1(A[j,i]);       /* Conste computacional: 1 FLOP  */
        b[j]= funcion2(A[j,i]);            /* Coste computacional: 1 FLOP   */
        A[j, i] = 0.0;                     /* Coste computacional: 1 FLOP   */
    }
```

Pregunta 1. ¿Qué hace? (no cómo lo hace, simplemente que obtiene).
Pregunta 2. Expresión analítica del tiempo de ejecución. Simplifique los cálculo usando aproximaciones tales como n por $(n-1)$ o $(n-2)$, i por $(i+1)$, etc.
Pregunta 3. Sea una descomposición del dominio. Dibuje el grafo de tareas resultante para $n = 5$.
Pregunta 4. No hay nxn procesadores. ¿Cuál es el mejor agrupamiento? ¿Por qué?
Pregunta 5. ¿El algoritmo paralelo resultante es balanceado? Razone sin realizar cálculos.
Pregunta 6. Análisis del rendimiento del algoritmo seleccionado, escalabilidad incluida.
Pregunta 7. ¿El algoritmo es escalable? Razone la respuesta. ¿Se podría haber llegado a la misma conclusión directamente desde la Pregunta 3? Razone la respuesta.

Solución

Pregunta 1. Pone a 0 los elementos de A por debajo de la diagonal principal. El resto de elementos de A y los del vector b los actualiza mediante las funciones *funcion1* y *funcion2*.

Pregunta 2

$$T(n) = \sum_{i=0}^{n-2} \sum_{j=n-1}^{i+1} \left[2 + \sum_{k=i+1}^{n-1} 1 \right]$$

equivalente a

$$T(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \left[2 + \sum_{k=i+1}^{n-1} 1 \right]$$

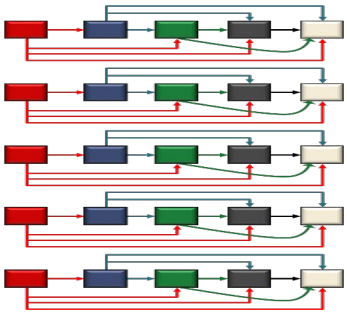
Siendo el resultado

$$T(n) = \frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6} \approx \frac{n^3}{3} + \frac{n^2}{2}$$

Si se usan las aproximaciones recomendadas en el enunciado de la pregunta entonces

$$T(n) \approx \frac{n^3}{3} + \frac{3n^2}{2} + \frac{7n}{6} \approx \frac{n^3}{3} + \frac{3n^2}{2}$$

Pregunta 3. El siguiente grafo muestra, usando código de colores, las 25 tareas correspondientes a los $5x5$ elementos de la matriz A. Nótese que no se representa el vector b porque no se pide. En cualquier caso, y sin pérdida de generalizar, el vector b se puede considerar como una columna más de A con lo que el grafo sería equivalente (una tarea más por fila afectada de igual forma, mismas dependencias, que el último elemento de cada fila de A).



Pregunta 4

Por filas o bloques de filas. Estos particionados anulan todas las comunicaciones/dependencias entre procesadores (procesos o hilos). Cuando el número de filas no es divisible entre el número de procesadores o, de forma general, un reparto cíclico por filas minimiza la pérdida de rendimiento pudiéndose considerar despreciable. Para el resto de preguntas se considerará que n es divisible por p y/o un particionado cíclico por filas adecuado.

Pregunta 5

Si el agrupamiento es por bloques de filas consecutivas el algoritmo resultante no es balanceado, dado que el proceso al que se asignan las n/p primeras filas realiza menos operaciones que el resto, y al que se le asignan las últimas n/p filas el que más. Con un particionado cíclico adecuado (al proceso con la primera fila se le asigna la última, al proceso con la segunda fila la penúltima, etc.) se minimizaría el desbalanceado de la carga pudiéndose considerar que el resultado es, en global, balanceado. Con un cíclico estándar (*round-robin*) habría desbalanceo de la carga, aunque en menor grado que por bloques de filas consecutivas.

Pregunta 6

Sea $T(n)$ el tiempo del algoritmo secuencial. En el particionado seleccionado (cíclico por filas adecuado) no hay comunicaciones/dependencias entre los procesadores y como este particionado divide balanceadamente la carga entre los procesadores entonces el tiempo paralelo ($T(n,p)$) es $T(n,p) = T(n)/p$.

$S(n,p)$=p, $E(n,p) = 1$, $C(n,p) = T(n)$ y $T_0(n,p)$=0. Como $T_0(n,p)$=0 el algoritmo es escalable.

Pregunta 7

Sí, al ver que no hay dependencias, que el algoritmo es balanceado y que el tamaño del problema se puede repartir sin restricciones entre un número concreto de procesadores.

Problema 6

Sean dos matrices A y B cuadradas de dimensiones nxn. Se desea resolver el siguiente problema "que todos y cada uno de los elementos de la matriz B al final valgan $B[i,j] = \sum_{r=1}^{n} A[i,r]B[i,r]$ ". Realizar el diseño completo para memoria distribuida basándose en descomposición del dominio. Razone todos y cada uno de los pasos (decisiones que se tomen).

Pregunta 1. Pseudocódigo del algoritmo secuencial y expresión analítica del tiempo de ejecución.

Pregunta 2. ¿Cuál o cuáles estructuras son relevantes para el particionado?

Pregunta 3. Construir el grafo de dependencias para un valor de n "manejable.

Pregunta 4. Pseudocódigo del algoritmo paralelo, tiempo de ejecución y eficiencia.

Pregunta 5. No hay nxn procesadores. ¿Cuál es el mejor agrupamiento? ¿Por qué?

Pregunta 6. Función de sobrecarga y análisis de la escalabilidad. ¿Es escalable?

Pregunta 7. Suponga que el problema fuese $B[i,j] = \sum_{r=1}^{n} A[r,j]B[i,r]$. ¿El diseño valdría con un número de cambios razonables? ¿Cuáles serían?

Solución

Pregunta 1

```
for (i=0; i<n; i++)
{
  tmp = 0.0;
  for (j=0; j<n; j++)
    tmp += B[i, j] * A[i, j];
  for (j=0; j<n; j++)
    B[i, j] = tmp;
}
```

$$T(n) = n(1 + 3n)t_c \approx 3n^2 t_c$$

Pregunta 2

Tanto A como B, cuadradas y de iguales dimensiones, intervienen en el problema pero A no se modifica con lo que no refleja el estado/evolución del problema. En estas condiciones la estructura relevante es B.

Pregunta 3

Hay varios grafos resultantes al considerar nxn procesadores cada uno de ellos con un solo elemento de A y B.

Uno de ellos, el más directo e intuitivo, está basado en la naturaleza de la operación a realizar: una reducción. La parte izquierda de la Figura 9 muestra el grafo de dependencias para una fila genérica de procesadores, supuesto que $n = 8$. Sobre el grafo se aplica la técnica "replicar comunicaciones / computación" ya que el resultado es necesario en todos y cada uno de los procesadores, obteniéndose la parte derecha de la Figura 9. Los fundamentos teóricos aquí descritos se han explicado en clase y están en las transparencias usadas en clases CEX.

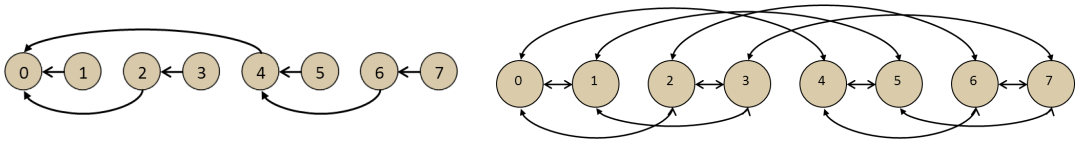


Figura 9. Grafo de dependencias del primer diseño (izquierda) y el mismo con la técnica de replicación (derecha)

No obstante, la metodología usada advierte que las comunicaciones globales conducen a algoritmos de bajo rendimiento y que se deben, siempre que sea posible, sustituir por operaciones "locales"; y cuando no sea factible esta sustitución proceder a un cambio global del diseño. Nuevamente esto se ha explicado en clase y está en las trasparencias de CEX.

Procediendo al cambio de enfoque, diseño, se sigue ahora otra de las recomendaciones dadas en clase "*dibuje los nodos del grafo (procesos) y desde el interior de uno de ellos observe su contexto y trátese de determinar su evolución temporal*". El grafo resultante, para $n = 4$, se muestra en Figura 10, donde se observa la existencia de un patrón: "*todos los procesos hacen lo mismo, que es, calcular, enviar a la izquierda, recibir de la derecha y así sucesivamente*". Nuevamente este patrón ha sido visto en varios ejemplos de las trasparencias de CEX.

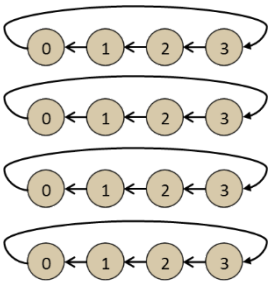


Figura 10. Grafo de dependencias del segundo diseño

Pregunta 4

La respuesta a esta pregunta parte del diseño de la Figura 10 que es el que utiliza comunicaciones locales y el que exhibe el mayor nivel de concurrencia porque las comunicaciones y cálculos de todos los procesos se solapan entre sí (comunicaciones con comunicaciones y cálculos con cálculos). El pseudocódigo y el análisis teórico es:

```
En procesador p(i, j)
  "b es B[i, j] y a es A[i, j]"
  tmp = b * a;
  for (i=0; i<n-1; i++)
  {
    Enviar a la izquierda b y a
    Recibir de la derecho en b y a
    tmp += b * a;
  }
  B[i, j]= tmp;
```

$$T(n, p = n) = (2n - 1)t_c + (n - 1)(2t_s + 4t_w) \approx 2n(t_c + t_s + 2t_w)$$

$$E(n, p = n) \approx \frac{3n^2 t_c}{2n^2(t_c + t_s + 2t_w)} = \frac{3t_c}{2(t_c + t_s + 2t_w)}$$

No obstante, para el primer diseño, Figura 9, el tiempo de ejecución es, aproximadamente,

$$T(n, p = n) \approx 2nt_c + (n - 1)(t_s + 2t_w) + (n - 1)(t_s + t_w) \approx 2n(t_c + t_s + 2t_w)$$

Que corresponde a la modelización de: 1) el cálculo del sumatorio de productos en un solo proceso que 2) recibe del resto (recolección) los datos y que 3) retorna el resultado al resto (difusión).

Pregunta 5

Por filas o bloques de filas. Elimina todas las comunicaciones.

Pregunta 6

Varios ejemplos ya hechos.

Pregunta 7

Sí, con hacer previamente la traspuesta de A, antes del reparto, sería suficiente.

Problema 7

Sea un equipo informático con una CPU con p núcleos y constante de cálculo t_c. El equipo tiene una GPU conectada al bus PCI-e con t_s la constante de establecimiento de canal y t_w la inversa del ancho de banda (constantes de comunicaciones del bus PCIe). Se sabe, además, que la potencia de cálculo de la GPU es 9x la de la CPU.

Se desea implementar una solución heterogénea (CPU y GPU colaboran computacionalmente) para un filtro gaussiano 3x3 con imágenes cuadradas de dimensiones $n\ x\ n, n > 0$. La aplicación del filtro tiene coste 1 *flop* por cada elemento de la imagen. Entonces:

Pregunta 1. ¿Qué cantidad de imagen asignaría a la CPU y la GPU? Obviamente, razonar la respuesta.

Pregunta 2. Expresión analítica del coste temporal secuencial, paralelo CPU y heterogéneo CPU/GPU para 1 iteración.

Pregunta 3. Supóngase que el coste de la carga de la imagen en la GPU y la recolección del resultado desde la GPU es 0, que $k = 1$ y que $p = 1$. ¿Cuál es la ganancia/pérdida de rendimiento de la solución heterogénea respecto a la paralela CPU?

Soución

Pregunta 1

Una parte a la CPU y 9 partes a la GPU. Más detalladamente, si el tamaño es n^2 y como la relación CPU *vs* GPU es 1:9, entonces $x + 9x = n^2$ sería el reparto óptimo, siendo x la cantidad de información asignada a la CPU. Consecuentemente, $x + 9x = n^2; 10x = n^2; x = \frac{n^2}{10}$ la cantidad de datos para la CPU y $\frac{9n^2}{10}$ para la GPU.

Remarcar que da igual que el reparto sea por bloques de filas o columnas consecutivas si el almacenamiento es 2D, al ser la matriz cuadrada. Si el almacenamiento fuese 1D se usaría reparto por bloques de filas consecutivas para *row-major* y por bloques de columnas consecutivas para *column-major*.

Pregunta 2

Número de elementos n^2, coste por iteración y elemento 1 *flop*. Entonces,

$T(n) = n^2 t_c$ Secuencial CPU

$T(n,p) = \frac{n^2 t_c}{p}$ Paralelo CPU

El tiempo heterogéneo se obtiene como sigue:

$$T_h(n,p) = T_{h_{enviar\_datos\_a\_GPU}}(n,p) + T_{h_{recibir\_resultados\_de\_GPU}}(n,p) + \max\left(T_{h_{computar\_iteracion\_en\_GPU}}(n,p), T_{h_{computar\_iteracion\_en\_CPU}}(n,p)\right) + T_{h_{intercambiar\_datos\_iteracion\_entre\_Gpu\_y\_Cpu}}(n,p)$$

Nótese que en el envío de la parte de la imagen necesaria a la GPU y la recolección del resultado (imagen tratada) de la GPU se mueve la misma información por lo que se puede simplificar la expresión multiplicando cualquiera de los términos por 2 y eliminando el otro. Además, con el reparto óptimo planteado ($\frac{n^2}{10}$ la cantidad de datos para la CPU y $\frac{9n^2}{10}$ para la GPU) se cumple (se puede formular la hipótesis) $T_{h_{computar\_iteracion\_en\_GPU}}(n,p) = T_{h_{computar\_iteracion\_en\_CPU}}(n,p)$ con lo que en la expresión del tiempo total se puede simplificar cambiando la expresión "max(…)" por el término más sencillo de los 2 anteriores. En resumen,

$$T_h(n,p) = 2T_{h_{enviar\_datos\_a\_GPU}}(n,p) + T_{h_{computar\_iteracion\_en\_CPU}}(n,p) + T_{h_{intercambiar\_datos\_iteracion\_entre\_Gpu\_y\_Cpu}}(n,p)$$

$$T_h(n,p) = 2\left(t_s + \frac{9n^2 t_w}{10}\right) + \frac{n^2 t_c}{10p} + T_{h_{intercambiar\_datos\_iteracion\_entre\_Gpu\_y\_Cpu}}(n,p)$$

Como es un gaussiano 3x3 el número de filas (o columnas según el tipo de reparto usado) que la GPU necesita de la CPU después de cada iteración para abordar la siguiente es 1, lo mismo que la CPU necesita de la GPU (la frontera "artificial" creada en el reparto). Consecuentemente,

$$T_h(n,p) = 2\left(t_s + \frac{9n^2 t_w}{10}\right) + \frac{n^2 t_c}{10p} + 2(t_s + nt_w)$$

El caso genera, cuando el número de iteraciones es k, es

$$T_h(n,p) = 2\left(t_s + \frac{9n^2 t_w}{10}\right) + k\left[\frac{n^2 t_c}{10p} + 2(t_s + nt_w)\right]$$

Pregunta 3

En las condiciones indicadas $T_h(n,p)$ es

$$T_h(n,p) = \frac{n^2}{10} t_c + 2(t_s + nt_w)$$

La ganancia/pérdida es $T(n,p) - T_h(n,p)$, esto es

$$n^2 t_c - \frac{n^2}{10} t_c - 2(t_s + nt_w) = 9n^2 t_c - 20(t_s + nt_w)$$

Habrá ganancia/pérdida según $9n^2 t_c$ sea mayor o menor que $20(t_s + nt_w)$. Se puede dejar así, aproximar, etc.

Problema 8

Sea un clúster formado por p equipo informático con una CPU mono-núcleos y constante de cálculo t_c. Los equipos están conectados por una red 10 Gigabit de alto rendimiento con un protocolo de comunicaciones ideal: el ancho de banda real es el teórico (no hay pérdidas de rendimiento).

El clúster es avanzado: la lectura de disco de las imágenes de entrada y la escritura en disco de las imágenes resultantes se hace en paralelo. En otras palabras, su coste temporal es igual que en el secuencial y no será considerado.

Se desea implementar una solución distribuida de un filtro gaussiano 3x3 para imágenes dimensiones $n\ x\ n, n > 0$ usando *char* como tipo base. La aplicación del filtro tiene coste 1 *flop* por cada elemento de la imagen por iteración. En estas condiciones se pide:

Pregunta 1. ¿Cuál es el valor de t_w?

Pregunta 2. Expresión analítica del coste temporal secuencial y paralelo para k iteraciones.

Pregunta 3. Dibujar la evolución de la eficiencia al mantener constante el tamaño de las imágenes y aumentar el número de procesadores. Sea muy preciso en la forma de la función. Razonar la respuesta.

Pregunta 4. ¿Qué forma deberían tener las imágenes para que la solución fuese claramente escalable?

Solución

Pregunta 1

t_w es la inversa del ancho de banda y como no hay pérdidas de rendimiento y el tipo base es *char* (igual a *byte*) su valor es $t_w = \frac{8}{10Gb} = 8^{-10}$ segundos por *byte*.

Pregunta 2

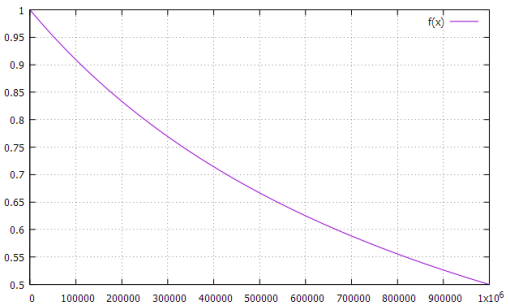
Para k repeticiones:

$T(n) = kn^2 t_c$ Secuencial

$T(n,p) = k\left(\frac{n^2 t_c}{p} + 2(t_s + nt_w)\right)$ Paralelo.

Pregunta 3

Si el tamaño es constante y p aumenta implica que el término $\frac{n^2 t_c}{p}$ disminuye y que $2(t_s + nt_w)$ permanece invariable. En términos analítica de la eficiencia sería, aproximadamente, $\frac{1}{1+\frac{p}{n}}$ y gráficamente



Pregunta 4

Rectangular. Caso extremo $n\ x\ 1$ porque de esta forma el peso de las comunicaciones sería mínimo.

Problema 9

Sea $A \in \mathcal{R}^{nxm}$ una matriz y p el número de ordenadores conectados por una red de datos con t_s la constante de establecimiento y t_w la inversa del ancho de banda. Cada ordenador tiene un procesador con k núcleos o *cores*, la constante de cálculo es t_c y el coste de las operaciones de sincronicación/comunicación internas es 1. Por simplicidad se asume que "n *es divisible entre* p".

La matriz A está distribuida entre los p ordenadores por "*bloques de filas consecutivas*": el ordenador p_0 tiene las $\frac{n}{p}$ primeras filas, el p_1 las siguientes $\frac{n}{p}$,…, y el último (p_{p-1}) las ultimas $\frac{n}{p}$ filas. Se pide, diseñar una función eficiente que calcule el valor promedio de los elementos de la matriz. Concretamente:

Pregunta 1. Complejidad espacial y temporal de la versión secuencial.

Pregunta 2. Dibujar la computación con rectángulos y las comunicaciones con flechas para $p = 4$, etiquetando cada elemento (rectángulo o flecha) con su peso o coste en términos de $n, m, p, k, t_s, t_w, t_c$.

Pregunta 3. Complejidad espacial y temporal de la versión paralela diseñada en 2º. Eficiencia, función de sobrecarga y escalabilidad.

Pregunta 4. Si el algoritmo fuese asíncrono y se cumpliese que $t_c = 10t_w = 10t_s$ y que $n = m$ ¿a qué conclusión se podría llegar?

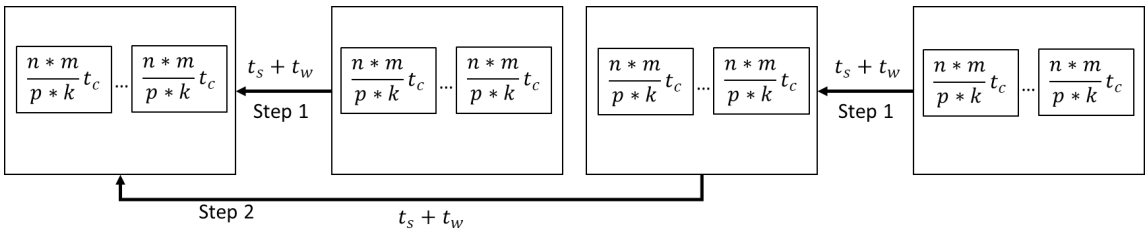
Solución

Pregunta 1

La espacial es el tamaño de la matriz, esto es, $\theta(n * m)$. La temporal del algoritmo secuencial es, aproximadamente, $T(n, m) = n * m * t_c \in \theta(n * m)$.

Pregunta 2

Se dibuja utilizando la representación usada en los apuntes de teoría, que es:



Pregunta 3

La espacial en cada procesador es $\frac{n*m}{p}$. En global es la misma que el secuencial, esto es, $\theta(n * m)$. La temporal del algoritmo paralelo es, aproximadamente, $T(n, m, p, k) = \frac{n*m}{p*k}t_c + \log(p) * (t_s + t_w)$.

$$E(n, m, p) = \frac{nmt_c}{p\left(\frac{nm}{pk}t_c + (t_s + t_w)\log(p)\right)} = \frac{t_c}{\frac{t_c}{k} + \frac{p\,(t_s + t_w)\log(p)}{nm}}$$

$$T_0(n, m, p) = \frac{nmt_c}{k} + p(t_s + t_w)\log(p) - nmt_c = \left(\frac{1}{k} - 1\right)nmt_c + p\,(t_s + t_w)\log(p)$$

Se sabe que en $T_0(n, m, p)$ el término en t_c no depende de p por lo que no afecta a la escalabilidad. Respecto a t_s y t_w su expresión es la misma por lo que con estudiar una de las constantes es suficiente. Por tanto

$$I: W \propto Kp\log(p) \in \theta(p\log(p))$$

Pregunta 4

Por un lado, $T(n, m, p, k) = \frac{n^2}{p*k}10t_w + 2t_w\log(p) = t_w\left(\frac{10n^2}{p*k} + 2\log(\mathrm{p})\right)$. Por el otro, que sea asíncrono no implica ventaja alguna puesto que todos los procesadores hacen, a la vez, los mismos cálculos, es decir, no habrá solapamiento entre cálculo y comunicaciones. En resumen, a ninguna conclusión que sea diferente a la que se puede plantear en el supuesto que sea síncrono.

Problema 10

En la separación Armónico-Percusivo intervienen estructuras de dimensiones $T_{MIDI}xT_{Real}$. Los alumnos de PCP han diseñado un algoritmo paralelo eficiente que resuelve el problema con un búfer circular usando una matriz de dimensiones mxn. El algoritmo es:

```
Inicializar adecuadamente la matriz AUDI de dimensiones mxn.
Repetir hasta final de la canción
        LeerDatos(AUDI, n)
        Procesar(AUDI, m, n)
        Distorsion(AUDI, m, n)
Fin repetir
```

Para validar el programa se usa un clúster con p ordenadores iguales conectados a una red, siendo t_s la constante de establecimiento y t_w la inversa del ancho de banda. Cada ordenador tiene una CPU con k núcleos, siendo la constante de cálculo t_c. Cada ordenador tiene la misma GPU de apoyo, siendo la GPU 9x más potente que la CPU. t_{sgpu} y t_{wgpu} son las constantes de las comunicaciones CPU/GPU. Por simplicidad se asume que "*n y m son divisibles entre p*". Después de pruebas se decide añadir un filtro gaussiano 1D, de alcance o tamaño 3, a todos los elementos de la matriz AUDI en el eje horizontal. Diseñar el nuevo módulo paralelo heterogéneo óptimo para el filtro respondiendo a las siguientes preguntas:

Pregunta 1. Selección del tipo de particionado, grafo de dependencias, tiempo secuencial y tiempo paralelo, suponiendo que hay mxn ordenadores y el coste por elemento del filtro es 1 *flop*.

Pregunta 2. Se debe agrupar. Seleccionar el mejor agrupamiento e indicar qué tipo de almacenamiento 1D más adecuado. ¿Qué cantidad de elementos de la matriz AUDI se debe asignar a cada procesador?

Pregunta 3. Olvídese del clúster y céntrese en un ordenador. Tiempo de ejecución secuencial y tiempo de ejecución paralelo heterogéneo.

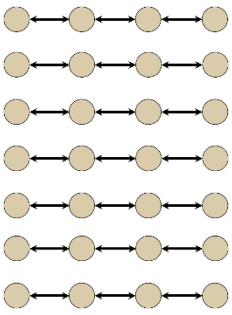
Pregunta 4. Con las expresiones de los tiempos de ejecución de 3) y suponiendo que las comunicaciones CPU/GPU tienen coste 0, plantar la expresión del tiempo de ejecución total (del clúster completo).

Solución

Pregunta 1

Problema similar al resuelto por los alumnos en las prácticas de la asignatura. Es, además, una simplificación del problema presentado y resuelto en clases presenciales de teoría. Por otra parte, es similar al problema 1, equivalente al problema 2 y una simplificación de los problemas 3 y 4 de este boletín de problemas. También es, algunos de sus apartados, el núcleo de los problemas 7 y 8.

Descomposición del Dominio. Ejemplo de grafo para $m = 7$ y $n = 4$.



P

or iteración

a) Secuencial: $T(m,n) = mnt_c$
b) Paralelo: $T(m,n,p = mn) = 2(t_s + t_w) + 2(t_s + t_w) + t_c$

Total, siendo R el número de iteraciones:

a) Secuencial: $T(m,n) = Rmnt_c$
b) paralelo: $T(m,n,p = mn) = R(2(t_s + t_w) + 2(t_s + t_w) + t_c)$

Pregunta 2

El mejor agrupamiento es por filas porque elimina todas las comunicaciones. Consecuentemente, el almacenamiento 1D más apropiado es por filas, da igual que sea por bloques consecutivos o cíclicos o … La cantidad de elementos por ordenador será: $\frac{mn}{p}$.

Pregunta 3

El tiempo de ejecución secuencial es el de la Pregunta 1, esto es, $T(m,n) = Rmnt_c$. Para el tiempo paralelo heterogéneo como la GPU es 9x más potente y el tamaño del problema por ordenador es $\frac{mn}{p}$ se tiene que $x + 9x = \frac{mn}{p}$ sería el reparto óptimo, siendo x la cantidad de información asignada a la CPU. Consecuentemente, $x + 9x = \frac{mn}{p}; 10x = \frac{mn}{p}; x = \frac{mn}{10p}$ es la cantidad de datos para la CPU y $\frac{9mn}{10p}$ para la GPU. La expresión del tiempo paralelo heterogéneo es:

$$T_h(m,n,p,k) = T_{h_{enviar\_datos\_a\_GPU}}(m,n,p,k) + T_{h_{recibir\_resultados\_de\_GPU}}(m,n,p,k) + \max\left(T_{h_{computar\_en\_GPU}}(m,n,p,k), T_{h_{computar\_en\_CPU}}(m,n,p,k)\right) + T_{h_{intercambiar\_datos\_Gpu\_y\_Cpu}}(m,n,p,k)$$

Nótese que en el envío de la parte de la matriz necesaria a la GPU y la recolección del resultado (parte de la matriz tratada) de la GPU se mueve la misma información por lo que se puede simplificar la expresión multiplicando cualquiera de los términos por 2 y eliminando el otro. Además, con el reparto óptimo planteado ($\frac{mn}{10p}$ la cantidad de datos para la CPU y $\frac{9mn}{10p}$ para la GPU) se cumple (se puede formular la hipótesis) que $T_{h_{computar\_en\_GPU}}(m,n,p) = T_{h_{computar\_en\_CPU}}(m,n,p)$, con lo que en la expresión del tiempo total se puede simplificar cambiando la expresión "max(…)" por el término más sencillo de los 2 anteriores. Finalmente, el término $T_{h_{intercambiar\_datos\_Gpu\_y\_Cpu}}(m,n,p)$ se sabe que es cero por el reparto por bloques de filas y la no existencia de comunicaciones (dependencias) entre elementos de distintas filas. En resumen,

$$T_h(m,n,p,k) = R\left(2T_{h_{enviar\_datos\_a\_GPU}}(m,n,p,k) + T_{h_{computar\_en\_CPU}}(m,n,p,k)\right) = R\left(2\left(t_{sgpu} + \frac{9mnt_{wgpu}}{10p}\right) + \frac{mnt_c}{10pk}\right),$$

donde k es el número de *cores* de la CPU.

Pregunta 4

Si las comunicaciones CPU/GPU tienen coste 0 el tiempo paralelo heterogéneo por ordenador es:

$$T_h(m,n,p,k) = R\left(\frac{mnt_c}{10pk}\right),$$

y el tiempo global del clúster

$$T(m,n,p,k) = 2p\left(t_s + \frac{mn}{p}t_w\right) + \frac{Rmnt_c}{10pk};$$

donde $\frac{mn}{p}$ es el reparto calculado en la Pregunta 2, el $2p$ proviene de considerar el envío, y la posterior recepción, de los datos del ordenador con la matriz AUDI completa al/del resto de ordenadores usando el modelado más básico, menos eficiente, para las operaciones de distribución propuesto en teoría. Entonces:

$$T_0(m,n,p,k) = pT(m,n,p,k) - T(m,n) = \frac{Rmnt_c}{10k} + 2p^2\left(t_s + \frac{mn}{p}t_w\right) - Rmnt_c = \left(\frac{1}{10k} - 1\right)Rmnt_c + 2p^2t_s + 2mnpt_w$$

Problema 11

Se dese construir un algoritmo paralelo heterogéneo óptimo para aplicar un filtro gaussiano 1D, con tamaño de máscara 3, a todos los elementos "*en el eje vertical*" de una matriz *M* de dimensiones mxn. Se ejecutará en un clúster con p ordenadores iguales conectados a una red (t_w es la inversa del ancho de banda y t_s la constante de establecimiento). Cada ordenador tiene una CPU mono-core, constante de cálculo es t_c, y una GPU. La GPU 19x más potente que la CPU. t_{sgpu} y t_{wgpu} son las constantes de las comunicaciones CPU/GPU. Por simplicidad se asume que "*n y m son divisibles entre p*". El filtro se debe aplicar R veces.

Pregunta 1. Selección del tipo de particionado, grafo de dependencias, tiempo secuencial y tiempo paralelo, suponiendo que hay mxn ordenadores y el coste por elemento del filtro es 1 *flop*.

Pregunta 2. Se debe agrupar. Seleccionar el mejor agrupamiento e indicar qué tipo de almacenamiento 1D más adecuado. ¿Qué cantidad de elementos de la matriz *M* se debe asignar a cada ordenador?

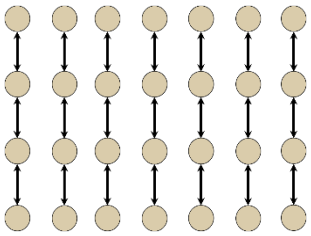
Pregunta 3. Olvídese del clúster y céntrese en un ordenador. Tiempo de ejecución secuencial y tiempo de ejecución paralelo heterogéneo.

Pregunta 4. Con las expresiones de los tiempos de ejecución de 3) y suponiendo que las comunicaciones CPU/GPU tienen coste 0, plantar la expresión del tiempo de ejecución total (del clúster completo). ¿Es escalable?

Solución

Pregunta 1

Descomposición del Dominio. Ejemplo de grafo para $m = 4$ y $n = 7$.



Por iteración

c) Secuencial: $T(m,n) = mnt_c$

d) Paralelo: $T(m,n,p = mn) = 2(t_s + t_w) + 2(t_s + t_w) + t_c$

Total, siendo R el número de iteraciones:

c) Secuencial: $T(m,n) = Rmnt_c$

d) paralelo: $T(m,n,p = mn) = R(4(t_s + t_w) + t_c)$

Pregunta 2

El mejor agrupamiento es por columnas porque elimina todas las comunicaciones. Consecuentemente, el almacenamiento 1D más apropiado es por columnas, da igual que sea por bloques consecutivos, cíclicos o … La cantidad de elementos por ordenador será: $\frac{mn}{p}$.

Pregunta 3

El tiempo de ejecución secuencial es el de la pregunta 1, esto es, $T(m,n) = Rmnt_c$. La GPU es 19x más potente por lo que $x + 19x = \frac{mn}{p}$ sería el reparto óptimo, siendo x la cantidad de información asignada a la CPU. Consecuentemente, $20x = \frac{mn}{p}$; $x = \frac{mn}{20p}$ es la cantidad de datos para la CPU y $\frac{19mn}{20p}$ para la GPU.

C) La expresión del tiempo paralelo heterogéneo es:

$$T_h(m,n,p) = T_{h_{enviar\_datos\_a\_GPU}}(m,n,p) + T_{h_{recibir\_resultados\_de\_GPU}}(m,n,p) + \max\left(T_{h_{computar\_en\_GPU}}(m,n,p), T_{h_{computar\_en\_CPU}}(m,n,p)\right) + T_{h_{intercambiar\_datos\_Gpu\_y\_Cpu}}(m,n,p)$$

Nótese que en el envío de la parte de la matriz necesaria a la GPU y la recolección del resultado (parte de la matriz tratada) de la GPU se mueve la misma información por lo que se puede simplificar la expresión

multiplicando cualquiera de los términos por 2 y eliminando el otro. Además, en el reparto óptimo ($\frac{mn}{20p}$ la cantidad de datos para la CPU y $\frac{19mn}{20p}$ para la GPU) se debe cumplir (se puede formular la hipótesis) que $T_{h_{computar\_en\_GPU}}(m,n,p) = T_{h_{computar\_en\_CPU}}(m,n,p)$, con lo que en la expresión del tiempo total se puede simplificar cambiando la expresión "max(…)" por el término más sencillo de los 2 anteriores. Finalmente, el término $T_{h_{intercambiar\_datos\_Gpu\_y\_Cpu}}(m,n,p)$ se sabe que es cero por el reparto por bloques de columnas y la no existencia de comunicaciones (dependencias) entre elementos de distintas columnas. En resumen,

$$T_h(m,n,p) = R\left(2T_{h_{enviar\_datos\_a\_GPU}}(m,n,p) + T_{h_{computar\_en\_CPU}}(m,n,p)\right)$$
$$= R\left(2\left(t_{sgpu} + \frac{19mnt_{wgpu}}{20p}\right) + \frac{mnt_c}{20p}\right).$$

Pregunta 4

Si las comunicaciones CPU/GPU tienen coste 0 el tiempo paralelo heterogéneo por ordenador es

$$T_h(m,n,p) = R\left(\frac{mnt_c}{20p}\right),$$

y el tiempo global del clúster

$$T(m,n,p) = 2p\left(t_s + \frac{mn}{p}t_w\right) + \frac{Rmnt_c}{20p},$$

donde $\frac{mn}{p}$ es el reparto calculado en la pregunta 2, el $2p$ proviene de considerar el envío, y la posterior recepción, de los datos del ordenador con la matriz M completa al/del resto de ordenadores usando el modelado más básico, menos eficiente, para las operaciones de distribución propuesto en teoría. Entonces:

$$T_0(m,n,p) = pT(m,n,p) - T(m,n) = \frac{Rmnt_c}{20} + 2p^2\left(t_s + \frac{mn}{p}t_w\right) - Rmnt_c$$
$$= \left(\frac{1}{20} - 1\right)Rmnt_c + 2p^2t_s + 2mnpt_w$$

Para simplificar el análisis de la escalabilidad supongamos que $m = n$ y, por tanto, $W = n^2$ Entonces:

t_c: No influye en la escalabilidad al no depender de p.

t_s: $W \propto Kp^2 \in O(p^2)$

t_w: $W \propto Kn^2p; W \propto KWp\,; 1 \propto Kp$

El máximo es $O(p^2)$ y entonces $I(W,p) = O(p^2)/O(n^2)$

Problema 12

Sea un clúster con p ordenadores iguales conectados a una red, siendo t_s la constante de establecimiento y t_w la inversa del ancho de banda. Cada ordenador tiene una CPU con k núcleos, siendo t_c la constante de cálculo. Cada ordenador tiene la misma GPU de apoyo, siendo t_{cgpu}, t_{sgpu} y t_{wgpu} las constantes de cálculo y de comunicaciones CPU/GPU, respectivamente. La GPU es 9x más potente que la CPU.

Sea $T(n) = n^2 t_c$ el tiempo secuencial y un diseño paralelo sin dependencias externas donde todos los ordenadores realizan la misma computación. Para el supuesto anterior se pide

Pregunta 1. Tiempo de ejecución paralelo de cada ordenador.

Pregunta 2. Si el coste de las comunicaciones CPU/GPU es 0, tiempo de ejecución del sistema.

Pregunta 3. Análisis de la escalabilidad y eficiencia escalada.

Solución

Pregunta 1

p ordenadores iguales y hacen el mismo trabajo. El reparto óptimo por ordenador es $\frac{n^2}{p}$. Puesto que la GPU es 9x CPU el reparto óptimo CPU/GPU por ordenador será $\frac{n^2}{p} = x + 9x = 10x; x = \frac{n^2}{10p}$ para CPU y $\frac{9n^2}{10p}$ para la GPU.

$$T(n,p,k) = T_{enviar\_GPU}(n) + T_{recibir\_GPU}(n) + T_{intercambiar}(n) + \max\left(T_{ar\_GPU}(n,p,k), T_{ar\_CPU}(n,p,k)\right)$$

Se envía y recibe la misma cantidad de datos a/de la GPU, se sabe que no hay dependencias externas y el reparto es óptimo. Entonces:

$$T(n,p,k) = 2T_{co\_GPU}(n) + T_{ar\_CPU}(n,p,k) = 2\left(t_{sgpu} + \frac{9n^2}{10p}t_{wgpu}\right) + \frac{n^2 t_c}{10pk}$$

Pregunta 2

$$T(n,p,k) = 2p\left(t_s + \frac{n^2}{p}t_w\right) + \frac{n^2 t_c}{10pk}$$

donde el primer término viene de distribuir/recolectar los datos entre los ordenadores, usando el modelo de comunicaciones P2P no óptimo.

Pregunta 3

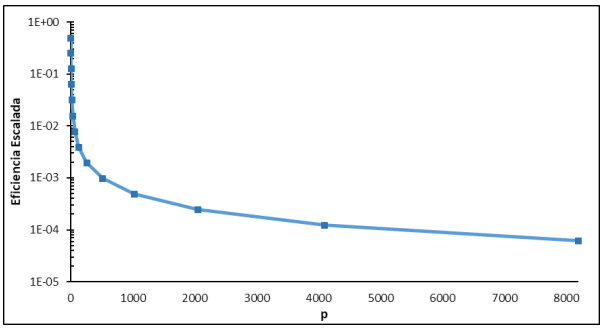
$$T_0(n,p,k) = pT(n,p,k) - T(n) = \frac{n^2 t_c}{10k} + 2p^2\left(t_s + \frac{n^2}{p}t_w\right) - n^2 t_c$$
$$= \left(\frac{1}{10k} - 1\right)n^2 t_c + 2p^2 t_s + 2n^2 p t_w$$

I_{t_c}: no depende de p, no influye en el análisis (ya visto).

$$I_{t_s}: W \propto Kp^2 \in O(p^2)$$
$$I_{t_w}: W \propto Kpn^2; 1 \propto Kp$$

Entonces $I(W,p) = \frac{O(pn^2)}{O(n^2)} \rightarrow$ no escala

Se ha supuesto $k = 4$ y todas las constantes con valor 1. Otros valores de k y/o para las constantes cambiarán la escala del eje Y, no la forma de la función.



La eficiencia no permanece constante lo que implica que el algoritmo no escala.

Problema 13. (Curso académico 18/19 – octubre 2018)
Sea un problema cuyo algoritmo secuencial óptimo vine caracterizado por $T(n) = n^2 t_c$. Sea un sistema informático compuesto por un ordenador central (NO CALCULA) y p ordenadores de cálculo, siendo $\frac{p}{2}$ del tipo A y el resto de tipo B. Todos los ordenadores están conectados una red de datos con t_s la constante de establecimiento y t_w la inversa del ancho de banda. Todos los equipos de cálculo tienen una constate de cálculo de valor t_c, los de tipo A son mono-core y los de tipo B dual-core. No hay dependencias ni comunicaciones durante la ejecución del algoritmo paralelo, a excepción de la distribución inicial de los datos del ordenador central a cada uno de los nodos de cálculo, que se debe considerar. En estas condiciones se pide:
Pregunta 1. Plantear la cantidad de información que debe recibir cada nodo de cálculo para que el algoritmo paralelo resultante sea óptimo.
Pregunta 2. Tiempo de ejecución paralelo, eficiencia, coste y función de sobrecarga.
Pregunta 3. Calcule las funciones de isoeficiencia y determine si el algoritmo paralelo es o no escalable.
Pregunta 4. Según la ley de Gustafson-Barsis el algoritmo es….

Solución
Pregunta 1
Los de tipo B son 2x los de tipo A, esto es, para un reparto óptimo cada ordenador de tipo B debe recibir el doble de trabajo que los de tipo A. Por otra parte, p_1(número de ordenadores de tipo A) y p_2(número de ordenadores de tipo B) cumplen que $p_1 = p_2 = \frac{p}{2}$. Suponiendo que el algoritmo secuencial es óptimo la carga computacional (W) es n^2 y, por tanto, W se debe dividir en $\frac{n^2}{p_1+p_2} = \frac{n^2}{p_1+2p_1} = \frac{n^2}{3p_1} = \frac{n^2}{\frac{3p}{2}} = \frac{2n^2}{3p}$ carga ajustada a los de tipo A, recibiendo los de tipo B el doble. Análogamente, $\frac{n^2}{p_1+p_2} = \frac{n^2}{\frac{p_2}{2}+p_2} = \frac{2n^2}{3p_2} = \frac{2n^2}{\frac{3p}{2}} = \frac{4n^2}{3p}$ carga ajustada a los de tipo B, recibiendo los de tipo A la mitad.
De otra forma (análoga a la resolución del problema de la CPU y la GPU en los apuntes de teoría).

$$Xp_1 + Yp_2 = n^2$$

Siendo X la cantidad de información que reciben los nodos de tipo A e Y la que recibien los de tipo B. Como $p_1 = p_2 = \frac{p}{2}$ y B es 2x A ($Y = 2X$) entonces

$$X\frac{p}{2} + 2X\frac{p}{2} = n^2 = \frac{3Xp}{2} \Rightarrow \frac{2n^2}{3p} = X$$

O como $X = Y/2$:

$$\frac{Y}{2}\frac{p}{2} + Y\frac{p}{2} = n^2 = \frac{3Yp}{4} \Rightarrow \frac{4n^2}{3p} = Y$$

Ejemplo: $p_1 = p_2 = 2;\ p_1 + p_2 = p = 4; n = 6 \Rightarrow n^2 = 36$. Entonces, $\frac{2n^2}{3p} = X = \frac{2*36}{3*4} = 6$ es la cantidad que reciben los de tipo A y 12 los de tipo B. O si se prefiere, $\frac{4n^2}{3p} = Y = \frac{4*36}{3*4} = 12$ es la cantidad que reciben los de tipo B y 6 los de tipo A.
Para continuar con el problema se debe indicar claramente qué modelo se usa, basado en X o Y. Se usa el primero ($\frac{2n^2}{3p}$), más general.
Nota: La diferencia con el ejercicio de teoría es que se refería al reparto interno (entre la CPU y la GPU; $p_1 = p_2 = 1$)

Pregunta 2
En primer lugar, analizaremos las comunicaciones. Cada ordenador de tipo A recibirá $p_1\left(t_s + \frac{2n^2 t_w}{3p}\right)$ y $p_2\left(t_s + \frac{4n^2 t_w}{3p}\right)$ los de tipo B. Nótese que una cosa es el número de ordenadores de cada tipo, variable que multiplica al factor entre los paréntesis, y otra la cantidad de datos que recibe cada uno de ellos, expresada en términos de lo calculado en el punto 1º. Operando sobre el tiempo de comunicaciones resulta:

$$p_1\left(t_s+\frac{2n^2t_w}{3p}\right)+p_2\left(t_s+\frac{4n^2t_w}{3p}\right)=\frac{p}{2}\left(t_s+\frac{2n^2t_w}{3p}\right)+\frac{p}{2}\left(t_s+\frac{4n^2t_w}{3p}\right)$$
$$=\frac{p}{2}t_s+\frac{p}{2}\frac{2n^2t_w}{3p}+\frac{p}{2}t_s+\frac{p}{2}\frac{4n^2t_w}{3p}=pt_s+\frac{n^2t_w}{3}+\frac{2n^2t_w}{3}=pt_s+n^2t_w$$

En consecuencia

$$T(n,p)=\frac{2n^2t_c}{3p}+pt_s+n^2t_w$$

$$E(n,p)=\frac{n^2t_c}{p\left(\frac{2n^2t_c}{3p}+pt_s+n^2t_w\right)}=\frac{n^2t_c}{\frac{2n^2t_c}{3}+p^2t_s+pn^2t_w}$$
$$E(n,p)=\frac{n^2t_c}{n^2\left(\frac{2t_c}{3}+pt_w\right)+p^2t_s}=\frac{t_c}{\left(\frac{2t_c}{3}+pt_w\right)+\frac{p^2t_s}{n^2}}$$

$$C(n,p)=pT(n,p)=p\left(\frac{2n^2t_c}{3p}+pt_s+n^2t_w\right)=\frac{2n^2t_c}{3}+p^2t_s+pn^2t_w$$

$$T_0(n,p)=pT(n,p)-T(n)=\frac{2n^2t_c}{3}+p^2t_s+pn^2t_w-n^2t_c=p^2t_s+pn^2t_w-\frac{n^2t_c}{3}$$

Pregunta 3

Partiendo de $T_0(n,p)$ la analizamos por partes (constantes). Se tiene que:

- $I_{t_c}: W \propto Kn^2$ que al no depender de p no influye en la escalabilidad.
- $I_{t_s}: W \propto Kp^2 \in O(p^2)$.
- $I_{t_w}: W \propto Kpn^2 = KpW$ el otro caso trivial analizado en los apuntes de teoría que concluye directamente que el algoritmo no es escalable.

Pregunta 4

Del punto 2 se tiene que

$$E(n,p)=\frac{t_c}{\left(\frac{2t_c}{3}+pt_w\right)+\frac{p^2t_s}{n^2}}$$

Entonces según Gustafson-Barsis

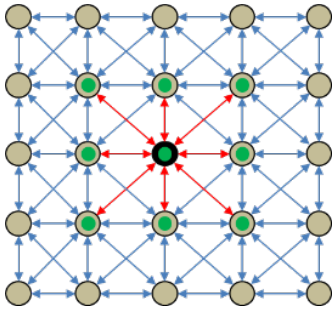
$$E(n,p)=c=\frac{t_c}{\left(\frac{2t_c}{3}+pt_w\right)+\frac{p^2t_s}{n^2}}\cong\frac{1}{\left(\frac{2}{3}+p\right)+\frac{p^2}{n^2}}$$
$$\frac{cp^2}{n^2}=1-\left(\frac{2}{3}+p\right)c$$
$$n^2=\frac{cp^2}{1-\left(\frac{2}{3}+p\right)c}=\frac{p^2}{\frac{1}{c}-\left(\frac{2}{3}+p\right)}\cong\frac{p^2}{\frac{2}{3}-p}\cong -p$$

Un crecimiento cuadrático de n precisa un crecimiento lineal en p. Apto para máquinas paralelas.

Problema 14. (Curso académico 18/19 – diciembre 2018)
Usando un sistema de memoria distribuida con p nodos, de constantes t_c, t_s y t_w, hay que diseñar un algoritmo paralelo para aplicar K veces un filtro gaussiano a imágenes de dimensiones NxN (cuadradas). El tamaño de la máscara de convolución es RxR y no se aprovecha la propiedad de la separabilidad. El número de *flop* para actualizar cada pixel de la imagen en cada iteración es proporcional al tamaño de la máscara (RxR flop más concretamente). Suponiendo que el coste de las comunicaciones de la distribución de la imagen original y la recolección de la imagen tratada es 0, se pide:
Pregunta 1. Ecuación del tiempo de ejecución del algoritmo secuencial.
Pregunta 2. Aplicar la metodología de diseño indicando el tipo de descomposición, dibujando el grafo para un caso concreto (por ejemplo, $R = 3$ y $N = 5$), indicando el tipo de comunicaciones y formulando la ecuación general del tiempo de ejecución paralelo para el diseño de granularidad mínima (mayor número de tareas).
Pregunta 3. Hay que agrupar. Demuestre, analíticamente, cuál o cuáles de los agrupamientos 1D es mejor.

Solución
Pregunta 1. $T(N,R) = KR^2N^2t_c$

Pregunta 2
Muy similar a los ejemplos de las transparencias CEX 72-75 y 90-97 del tema relacionado: particionado del dominio y comunicaciones locales. La figura adjunta es el grafo para los valores de R y N propuestos, resaltando en negro la tarea punto de observación, en rojo sus dependencias y en verde la máscara superpuesta (centrada en la tarea de observación).



La ecuación general del tiempo de ejecución paralelo para el diseño de granularidad mínima (mayor número de tareas) suponiendo que el intercambio de información entre 2 tareas vecinas (A envía a B y recibe de B; B recibe de A y envía a A) se contabiliza como una operación.

$$T(N,R,p) = K(R^2t_c + (R^2-1)(t_s+t_w)) \approx KR^2(t_c+t_s+t_w)$$

En este caso $p = N^2$.

Pregunta 3
Para la demostración se usará la Relación Superficie Volumen (RSV) de los particionados 1D por bloques de filas o columnas consecutivas. Nótese que en condiciones de uso ($N \gg p$, $R > 2$ y $p > R$) los modelos cíclicos añadirían más comunicaciones y como no hay desbalanceo en la computación son peores. Así, para el algoritmo del apartado 2:

$$RSV = K\frac{(R^2-1)(t_s+t_w)}{R^2t_c} \approx \frac{t_s+t_w}{t_c}$$

Agrupando por bloques de columnas consecutivas

$$RSV \approx \frac{2\left(t_s + \left(\frac{R-1}{2}\right)Nt_w\right)}{\frac{N^2R^2}{p}t_c} \approx \frac{2p\left(t_s + \frac{R}{2}Nt_w\right)}{N^2R^2t_c} = \frac{2pt_s + pRNt_w}{N^2R^2t_c}$$

Agrupando por bloques de filas consecutivas

$$RSV \approx \frac{2\left(t_s + \left(\frac{R-1}{2}\right)Nt_w\right)}{\frac{N^2R^2}{p}t_c} \approx \frac{2p\left(t_s + \frac{R}{2}Nt_w\right)}{N^2R^2t_c} = \frac{2pt_s + pRNt_w}{N^2R^2t_c}$$

Ambos agrupamientos son iguales, lo que es lógico porque las imágenes a tratar son cuadradas.

Problema 15. (Curso académico 18/19 – diciembre 2018)
A la vista de las ecuaciones del tiempo de ejecución paralelo obtenidas en el problema anterior la empresa considera que el uso de un sistema paralelo de memoria distribuida no es apropiado, máxime si se tiene en consideración que los tiempos de distribución/recolección inicial/final no son 0, por lo que desea explorar el comportamiento en un único ordenador con una GPU.
El ordenador tiene una CPU con p núcleos y constante de cálculo t_c. La GPU tiene una potencia de cálculo $\boldsymbol{S}x$ la de la CPU, siendo t_s y t_w las constantes de comunicación entre la CPU y la GPU. Se pide:
Pregunta 1. Cantidad de imagen asignaría a la CPU y la GPU. Razonar analíticamente la respuesta.
Pregunta 2. Expresión analítica del coste temporal paralelo CPU y heterogéneo CPU/GPU.
Pregunta 3. Suponga que $K = 1$, que $t_c = t_s = t_W = 1$, que $p = R$ y que el coste de la carga/recolección de la imagen/resultado en/de la GPU es 0 ¿Qué potencia debería tener la GPU para que la versión heterogénea fuese recomendable?

Solución
Pregunta 1
Una parte a la CPU y S partes a la GPU. Más detalladamente, si el tamaño es N^2 y como la relación CPU *vs* GPU es $1:S$, entonces $x + Sx = N^2$ sería el reparto óptimo, siendo x la cantidad asignada a la CPU. Consecuentemente, $x + Sx = N^2; (S+1)x = N^2; x = \frac{N^2}{S+1}$ la cantidad de datos para la CPU y $\frac{SN^2}{S+1}$ para la GPU.
Remarcar que da igual que el reparto sea por bloques de filas o columnas consecutivas si el almacenamiento es 2D, al ser la matriz cuadrada. Si el almacenamiento fuese 1D se usaría reparto por bloques de filas consecutivas para *row-major* y por bloques de columnas consecutivas para *column-major*.

Pregunta 2
Número de elementos N^2, coste por iteración y elemento R^2 *flop*. Entonces,

$T(N,R) = KR^2N^2t_c$	Secuencial CPU, del problema anterior.
$T(N,R,p) = \frac{KR^2N^2t_c}{p}$	Paralelo CPU.

El tiempo heterogéneo se obtiene como sigue:

$$T_h(N,R,p) = T_{h_{enviar\_datos\_a\_GPU}}(N,R,p) + T_{h_{recibir\_resultados\_de\_GPU}}(N,R,p) + \max\left(T_{h_{computar\_iteracion\_en\_GPU}}(N,R,p), T_{h_{computar\_iteracion\_en\_CPU}}(N,R,p)\right) + T_{h_{intercambiar\_datos\_iteracion\_entre\_Gpu\_y\_Cpu}}(N,R,p)$$

En el envío de la parte de la imagen necesaria a la GPU y la recolección del resultado de la GPU se mueve la misma información por lo que se puede simplificar la expresión multiplicando cualquiera de los términos por 2 y eliminando el otro. Además, con el reparto óptimo planteado se puede formular la hipótesis: $T_{h_{computar\_iteracion\_en\_GPU}}(N,R,p) = T_{h_{computar\_iteracion\_en\_CPU}}(N,R,p)$ con lo que en la expresión del tiempo total se puede simplificar cambiando la expresión "max(…)" por el término más sencillo de los 2 anteriores. En resumen,

$$T_h(N,R,p) = 2T_{h_{enviar\_datos\_a\_GPU}}(N,R,p) + T_{h_{computar\_iteracion\_en\_CPU}}(N,R,p) + T_{h_{intercambiar\_datos\_iteracion\_entre\_Gpu\_y\_Cpu}}(N,R,p)$$

$$T_h(N,R,p) = 2\left(t_s + \frac{SN^2t_w}{S+1}\right) + \frac{N^2R^2t_c}{(S+1)p} + T_{h_{intercambiar\_datos\_iteracion\_entre\_Gpu\_y\_Cpu}}(N,R,p)$$

Como es un gaussiano de RxR el número de filas (o columnas según el tipo de reparto usado) que la GPU necesita de la CPU después de cada iteración para abordar la siguiente es $\frac{R-1}{2}$, lo mismo que la CPU necesita de la GPU (la frontera "artificial" creada en el reparto). Consecuentemente,

$$T_h(N,R,p) = 2\left(t_s + \frac{SN^2t_w}{S+1}\right) + \frac{N^2R^2t_c}{(S+1)p} + 2\left(t_s + \frac{R-1}{2}Nt_w\right)$$
$$\approx\ 2\left(t_s + \frac{SN^2t_w}{S}\right) + \frac{N^2R^2t_c}{Sp} + 2\left(t_s + \frac{RNt_w}{2}\right)$$

El caso genera, cuando el número de iteraciones es K, es

$$T_h(N,R,p) \approx\ 2\left(t_s + \frac{SN^2t_w}{S}\right) + \left[\frac{N^2R^2t_c}{Sp} + 2\left(t_s + \frac{RNt_w}{2}\right)\right]K$$

Pregunta 3
Al suponer $K = 1$, $t_c = t_s = t_W = 1$, $p = R$ y 0 ciertas comunicaciones se tiene que

$$T_h(N,R,p) \approx\ \frac{N^2p}{S} + 2\left(1 + \frac{pN}{2}\right) \approx \frac{N^2p}{S} + pN$$

y que el tiempo paralelo CPU es

$$T(N,R,p) = pN^2$$

Para que la versión heterogénea sea interesante se debe cumplir que $T(N,R,p) - T_h(N,R,p) > 0$ por lo que

$$pN^2 - \left(\frac{N^2p}{S} + pN\right) = pN(N-1) - \frac{N^2p}{S} > 0 \Rightarrow\ pN(N-1) > \frac{N^2p}{S}$$

$$S > \frac{N^2p}{pN(N-1)} = \frac{N}{N-1}$$

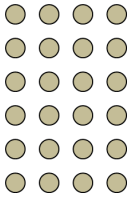
La respuesta es: cualquier valor de S mayor que $\frac{N}{N-1}$. Nótese que por la naturaleza del problema $N \geq 3$ (R es por definición mayor o igual que 3) por lo que la expresión $\frac{N}{N-1}$ siempre está definida (no hay división por 0). Entonces, cuando N es muy pequeño con $S = 2$ se garantiza que la versión heterogénea es apropiada. Cuando N es grande (valores estándar) con $S = 1$ se podría afirmar que también la versión heterogénea es apropiada.

Problema 16. (Curso académico 18/19 – mayo 2019)
Sean A y B dos matrices, de dimensiones $m \times n$, de números complejos $(\alpha + \beta i)$. Sea p el número de procesadores que se pueden configurar tanto en línea como en una malla bidimensional no cerrada. p tiene raíz exacta y las dimensiones son divisibles tanto por p como por $\sqrt{p}$.
El problema consiste en obtener C, de dimensiones $m \times n$, tal que $C_{i,j} = A_{i,j} \oplus B_{i,j} \ \forall_{1 \le i \le m, 1 \le j \le n}$, donde $\oplus$ denota la multiplicación de números complejos, que se define como

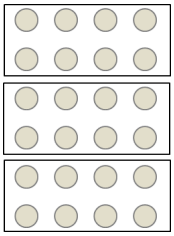
$$(a_1 + b_1 i) \oplus (a_2 + b_2 i) = (a_1 a_2 - b_1 b_2) + (a_1 b_2 + b_1 a_2) i$$

Por convenio se asume que el producto de dos números complejos son 6 *flop*.
El problema será resuelto en un sistema paralelo. Son varios los particionados 1D / 2D los posibles. El alumno debe seleccionar el mejor particionado justificando su elección mediante la realización de un estudio teórico completo. Realizar el análisis de la escalabilidad. Dibuje la eficiencia escalada del diseño.

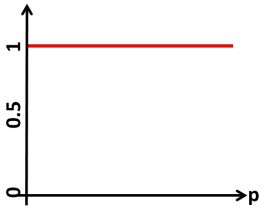
Solución
Como se ha visto en clase, la mejor opción es realizad un particionado del dominio. Considerando las indicaciones dadas en el enunciado el grafo resultante es el que se muestra en la figura, donde se observa que no hay dependencias/comunicaciones, y que todas las tareas realizan simultáneamente la misma computación.



Partiendo de las consideraciones anteriores cualquier particionado 1D/2D sobre los ejes dan un resultado óptimo (ante la duda, aplicar la relación superficie-volumen). Se elige, por simplicidad, el particionado 1D sobre el eje de mayor dimensión. Sea este el eje *Y*, esto es, se está suponiendo que $m \geq n$. El resultado, para el supuesto de $p = 3$, es el que se muestra en la figura.



$T(m \times n, 1) = 6mnt_c$
Al no haber dependencias, el reparto ser balanceado y trabajar todos los procesadores concurrentemente se tiene que
$T(m \times n, p) = \frac{6mn}{p} t_c$. $E(m \times n, p) = 1$. $S(m \times n, p) = p$.
$C(m \times n, p) = pT(m \times n, p) = T(m \times n, 1)$
La función de sobrecarga $T_0(m \times n, p) = C(m \times n, p) - T(m \times n, 1) = 0$ y, por consiguiente, el diseño es escalable.
La eficiencia escalada, al ser la eficiencia constante con valor 1, es



Problema 17. (Curso académico 18/19 – junio 2019)
Sean A y B dos matrices, de dimensiones $m \times n$ con $m > n$, de números complejos $(\alpha + \beta i)$. El problema consiste en, iterativamente, actualizar C tal que $C_{i,j} = C_{i,j} + A_{i,j} \oplus B_{i,j}\ \forall_{1 \le i \le m, 1 \le j \le n}$, donde $\oplus$ se define como

$$(a_1 + b_1 i) \oplus (a_2 + b_2 i) = (a_1 a_2 - b_1 b_2) + (a_1 b_2 + b_1 a_2) i$$

El número de repeticiones depende de que, al final de cada iteración, el $\frac{\sqrt{\|\sum C_{i,j}\|_2}}{\sqrt{m*n}} \forall_{1 \le i \le m, 1 \le j \le n}$ de la parte real de C sea inferior a una determinada cota de error ε.
El problema será resuelto en un sistema paralelo con p procesadores, que se pueden configurar tanto en línea como en una malla bidimensional no cerrada. p tiene raíz exacta y las dimensiones son divisibles tanto por p como por $\sqrt{p}$. Por convenio se asume que el producto de dos números complejos son 6 *flop*.

Son varios los particionados 1D / 2D posibles. El alumno debe seleccionar el mejor particionado justificando su elección mediante la realización de un estudio teórico completo. Realizar el análisis de la escalabilidad.
Para uniformizar y simplificar el diseño suponer: que a) $m = 4$ y $n = 3$, b) que el número de iteraciones es k, y c) que no se exige que el diseño sea óptimo.

Solución
Como se ha visto en clase, la mejor opción es realizad un particionado del dominio sobre la matriz C. En la primera etapa del diseño se asume que se tienen tantos procesadores como elementos tiene el dominio, esto es, $p = m \times n$. El problema a resolver tiene, en cada iteración, 2 partes bien diferenciadas. La primera corresponde con la actualización de los elementos de la matriz C, donde no hay dependencias/comunicaciones y todas las tareas realizan simultáneamente la misma computación (ver Figura 11 Izquierda). Por tanto, el tiempo paralelo por iteración de esta fase es

$$T_1\left(m \times n, p_{con\ p=m \times n}\right) = 6t_c$$

La segunda es la aplicación de un operador de reducción. No se exige que su diseño sea óptimo, por tanto una solución (menos eficiente) tomada directamente de los apuntes CEX se muestra en el centro de Figura 11. Esta solución está basada en comunicaciones globales de todos a uno (tarea remarcada en negro) que, una vez recibidos todos los datos (los elementos de C) realiza la reducción (sumatorio). El coste de esta operación sabiendo que $p = m \times n$ es

$$T_2\left(m \times n, p_{con\ p=m \times n}\right) = (p-1)(t_s + t_w) + mnt_c \cong mn(t_s + t_w) + mnt_c$$

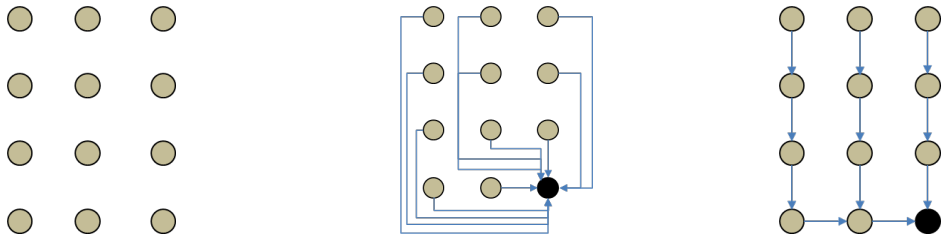


Figura 11. Izquierda, 1ª fase del diseño. Centro, incluyendo la 2ª fase más ineficiente. Derecha, con una 2ª fase más óptima

También se muestra en Figura 11, a la derecha, un diseño más eficiente, sin llegar al modelo "en árbol", que primero calcula las sumas parciales por columnas y luego la suma total. En este modelo, la suma de las columnas se hace en paralelo y, en cada columna, se va propagando la suma desde la primera fila a la última. Calculadas las sumas parciales de las columnas, se obtiene la final propagando la suma de las primeras columnas a la última. Nótese que, dentro de cada columna, y en la fila final, no se pueden solapar los cálculos y comunicaciones. Entonces se tiene que el tiempo total por columna será

$$T_{2_{col}}\left(m \times n, p_{con\ p=m \times n}\right) = (m-1)[(t_s + t_w) + t_c] \cong m[t_s + t_w + t_c]$$

Siguiendo el mismo razonamiento para la suma final, que no se solapa con las anteriores, se tiene

$$T_{2_{fil}}\left(m \times n, p_{con\ p=m\times n}\right) = (n-1)[(t_s + t_w) + t_c] \cong n[t_s + t_w + t_c]$$

Finalmente, el tiempo es

$$T_2\left(m \times n, p_{con\ p=m\times n}\right) \cong n[(t_s + t_w) + t_c] + m[(t_s + t_w) + t_c] = (n+m)[t_s + t_w + t_c]$$

que como se puede apreciar en inferior al T_2 del diseño anterior. Optando por este último diseño, el tiempo total paralelo es

$$T\left(m \times n, p_{con\ p=m\times n}\right) \cong (n+m)(t_s + t_w) + 6t_c$$

Por otra parte, el tiempo secuencial por iteración es

$$T(m \times n, 1) = (6+1)mnt_c \cong 6mnt_c$$

donde con la constante 1 se denota el coste adicional para obtener la suma de los elementos de C. En Figura 12 se muestran varios agrupamientos 1D/2D para el diseño paralelo elegido. En 1D el número de comuniones por iteración es $(p-1)$, independientemente de que sea por filas o columnas. En 2D también es $(p-1)$, pero el modelo de implementación resulta menos simple. En resumen, todos ellos tienen la misma complejidad por lo que se opta por uno de los 1D. A una conclusión similar se llega aplicando la relación superficie-volumen. En consecuencia,

$$T(m \times n, p) = \frac{(6+1)mnt_c}{p} t_c + (p-1)(t_s + t_w) \cong \frac{6mnt_c}{p} + p(t_s + t_w)$$

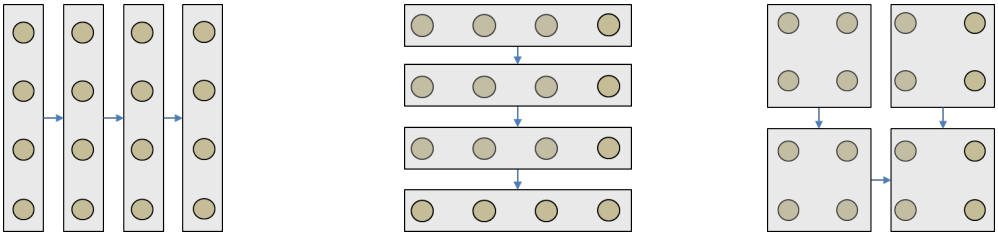


Figura 12. Agrupamientos 1D/2D para el diseño paralelo realizado

La eficiencia

$$E(m \times n, p) = \frac{6mnt_c}{p\left(\frac{6mn}{p} t_c + p(t_s + t_w)\right)} = \frac{6mnt_c}{6mnt_c + p^2(t_s + t_w)}$$

El Coste

$$C(m \times n, p) = p\left(\frac{6mn}{p} t_c + p(t_s + t_w)\right) = 6mnt_c + p^2(t_s + t_w)$$

La función de sobrecarga

$$T_0(m \times n, p) = 6mnt_c + p^2(t_s + t_w) - 6mnt_c = p^2(t_s + t_w) = p^2 t_s + p^2 t_w$$

Y el análisis de la escalabilidad, suponiendo por simplicidad que $m = n$ y sabiendo que el estudio es el mismo para t_s y t_w,

$$I_w = I_{t_s}: W \propto Kp^2 \in O(p^2)$$

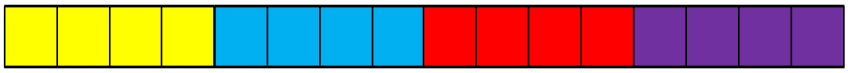
Problema 18 (Curso académico 19/20, diciembre 2019)
Sea M una matriz cuadrada de dimensiones $32x32$, con almacenamiento 1D r*ow-major*, y FPA un número $\epsilon\mathbb{R}$. Sea un *kernel* que se diseña considerando que siempre será ejecutado como "xxx<<<NB, NT>>>(M, FPA, 32)", siendo "dim3 $NB(1,1,1)$" y "dim3 $NT(32,1,1)$". El tratamiento que se aplica a cada elemento de la matriz es "si $(M[\mathrm{i}] < \mathrm{FPA})\ M[\mathrm{i}] = 0.0$". Sea el siguiente *kernel*:

```
__global__ void Caso1(double *M, const double FPA, const int N) {
  for(int i=0; i<N; i++)
    if(M[threadIdx.x*N + i] < FPA) M[threadIdx.x*N + i]=0.0;
}
```

Se pide

1. ¿Es coalescente? Razone la respuesta apoyándose en un dibujo suponiendo que M es de 4x4 (dibujo con formato 1D *row-major*) y que $NT.x = 4$.
2. Si es coalescente diseñar un *kernel* que no lo sea, nuevamente razonando la respuesta usando como apoyo un dibujo suponiendo que M es de 4x4 y que $NT.x = 4$. Si no es coalescente, diseñar un *kernel* que sí lo sea, nuevamente ….

Solución
Cada hilo trabaja sobre N posiciones de la matriz, con patrón de acceso $[threadIdx.x * N + \cdots]$, es decir, cada hilo actúa sobre una fila de la matriz. Dado que la matriz está almacenada 1D *row-major* los elementos que modifica cada hilo están consecutivos en memoria, pero **NO LO** están los elementos de igual columna y distinta fila. La figura adjunta muestra el patrón de acceso con código de colores: cada color identifica a uno de los 4 hilos.



Como hay N hilos ejecutando el mismo código, se está accediendo simultáneamente a N posiciones que están a distancia "32 x sizeof(double)", que es mayor que 128 Bytes. Consecuentemente no es coalescente. Esto es consecuencia de haber diseñado el *kernel* tomando como referencia lo que se hace con OpenMP.

Por todo lo razonado el *kernel* coalescente sería:

```
__global__ void Caso2(double *M, const double FPA, const int N) {
  for(int i=0; i<N; i++)
    if(M[threadIdx.x + N*i] < FPA) M[threadIdx.x + N*i]=0.0;
}
```

cuyo patrón de acceso se muestra en la siguiente figura

Problema 19 (Curso académico 19/20, octubre 2019)
Sea un sistema informático compuesto por p ordenadores conectados a una red de datos de constantes t_s y t_w. Los *p* equipos son iguales y t_c es la constante cálculo. La red de datos solo tiene una limitación: un equipo no puede participar simultáneamente en 2 o más operaciones P2P.
El problema a resolver usa como datos de entrada una estructura bidimensional que, por simplicidad, se supondrá cuadrada (n^2). El algoritmo secuencial consiste en:

1. Realizar una transformación sobre los datos de entrada de intensidad computacional (*W*) cúbica. El resultado será una estructura unidimensional de tamaño *n*.
2. Sobre el resultado de 1. se aplica un nuevo proceso, de intensidad computacional cuadrática.

Los programadores han diseñado un algoritmo paralelo que consiste en:

I. Repartir la computación de 1. entre los *p* equipos, de forma balanceada, sin ningún tipo de dependencia y/o sobrecarga. Al final cada equipo tendrá una *p-ava* parte de la estructura unidimensional resultante.
II. Intercambiar las *p-avas* partes para que todos los ordenadores tengan completa el resultado de 1.
III. Cada ordenado aplica el paso 2. en las mismas condiciones que I. (sin dependencias / sobrecargas).

Dada la naturaleza de la red de datos, los programadores dudan entre 2 posibles alternativas para intercambiar las *p-avas* partes, estas son:

1. Todos los equipos envían su parte a uno y éste, una vez conformada la estructura unidimensional de tamaño *n*, la envía al resto.
2. Todos los equipos envían su parte al resto y recibe, del resto, las *p-1* partes (*todos con todos*).

Se pide:

A. Ecuación del tiempo de ejecución del algoritmo secuencial (0.25 puntos).
B. Modelar los tiempos de comunicación de las 2 alternativas. Razonar cuál es la mejor (1.25 puntos).
C. Ecuación del tiempo de ejecución del algoritmo paralelo (0.5 puntos).
D. Coste (0.5 puntos), eficiencia (0.5 puntos), sobrecarga (0.5 puntos)
E. Análisis de escalabilidad / isoeficiencia y conclusión sobre la escalabilidad del algoritmo (1.5 puntos).

Nota. Las aproximaciones $(n-1)=n$, $(p-1)=p$, etc. son admitas.

Solución

A. $T(n) = n^3 t_c + n^2 t_c = (n^3 + n^2) t_c$

B. Alternativa 1.: $T(n,p) = (p-1)\left(t_s + \frac{n}{p} t_w\right) + (p-1)(t_s + n t_w) \cong p\left(t_s + \frac{n}{p} t_w\right) + p(t_s + n t_w)$

   Alternativa 2: $T(n,p) = (p-1)\left(t_s + \frac{n}{p} t_w\right) + (p-1)\left(t_s + \frac{n}{p} t_w\right) \cong p\left(t_s + \frac{n}{p} t_w\right) + p\left(t_s + \frac{n}{p} t_w\right)$

   Ambas alternativas realizan el mismo tipo y número de comunicaciones ($\cong 2p$), pero la segunda envía menos datos $\left(\frac{n}{p}\ vs\ n\right)$ en p comunicaciones. Consecuentemente la segunda es la mejor.

C. $T(n,p) = \frac{n^3}{p} t_c + \frac{n^2}{p} t_c + p\left(t_s + \frac{n}{p} t_w\right) + p\left(t_s + \frac{n}{p} t_w\right) = (n^3 + n^2)\frac{t_c}{p} + 2p t_s + 2n t_w$

D. $C(n,p) = pT(n,p) = (n^3 + n^2) t_c + 2p(p t_s + n t_w)$

   $E(n,p) = \frac{T(n)}{C(n,p)} = \frac{(n^3+n^2)t_c}{(n^3+n^2)t_c + 2p(p t_s + n t_w)} = \frac{1}{1 + \frac{2p(p t_s + n t_w)}{(n^3+n^2)t_c}}$

   $T_0(n,p) = C(n,p) - T(n) = 2p^2 t_s + 2pn t_w$

E. Atendiendo a la expresión de $T_0(n,p)$

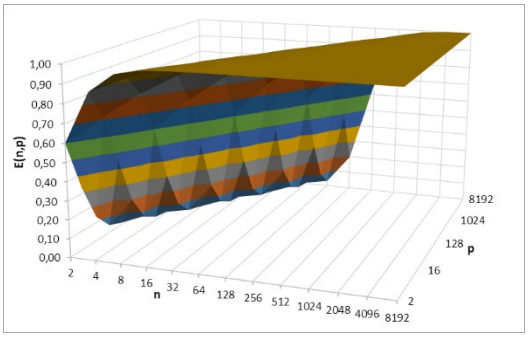
   $\mathrm{I_{t_s}}: \mathrm{W} \propto \mathrm{Kp^2} \in \mathrm{O(p^2)}$

$I_{t_w}: W \propto Kpn \cong KW^{\frac{1}{3}}p; W^{\frac{2}{3}} \propto Kp; W \propto (Kp)^{\frac{3}{2}} = Kp\sqrt{p} \in O(p\sqrt{p})$

Enfocando el análisis por la otra vía:

$$I(W,p) = \frac{pT(n,p) - T(n)}{T(n)} = \frac{O(p^2)}{O(n^3)}$$

Según la expresión anterior el algoritmo es escalable. La figura adjunta, no se pedía, corrobora empíricamente la afirmación.



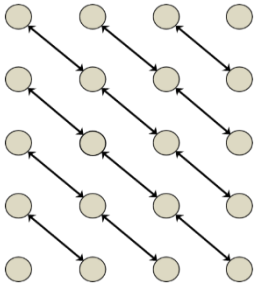
Nótese que la "aparente" contradicción viene de la simplificación del problema: no se están considerando las comunicaciones necesarias para distribuir la matriz de datos entre los procesadores, ni las pertinentes para recolectar el resultado final, en la expresión del tiempo de ejecución paralelo.

Problema 20 (Curso académico 19/20, diciembre 2019)
Sea una matriz un de dimensiones NxM con $N > M$. El problema consiste en iterar k veces sobre todos los elementos de la matriz, siendo el cálculo aplicado en cada iteración:

$$X[i,j]^{t+1} = \frac{4X[i,j]^t + X[i-1,j-1]^t + X[i+1,j+1]^t}{6}$$

Aplicar las fases previas a la "asignación" de la metodología "Fundamentos del Diseño de Algoritmos" para obtener un diseño óptimo. Usar $N = 5$ y $M = 4$ a la hora de representar/dibujar los grafos resultantes. Se pueden usar aproximaciones tales como x para $(x-1)$ o $(x-2)$, suponer *padding* para las fronteras naturales del grafo/matriz, etc.

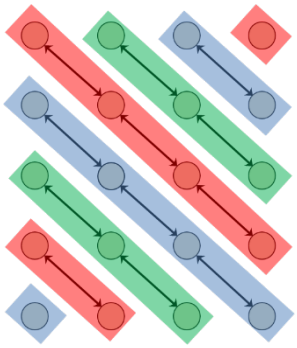
Solución
La solución es similar al problema resuelto en la transparencia 77 del tema de teoría correspondiente, siendo el dibujo resultante, cuando se suponen tantos procesadores como elementos tiene la matriz, el que se muestra en la figura adjunta.



El código paralelo correspondiente es similar al de la transparencia 82, al igual que las consideraciones sobre la concurrencia, balanceado, etc. La expresión del tiempo de ejecución es:

$$T(NM,p) = k(2(t_s + t_w) + 2(t_s + t_w) + 3t_c)$$

Para agrupar, fase 3ª del diseño, y última que se pide en la resolución del ejercicio, los resultados numéricos para la relación superficie/volumen cuando se aplica 1D por filas, 1D por columnas y 2D por bloques son equivalentes a los mostrados en la transparencia 100 y sucesivas. La única salvedad es que, como $N > M$, siempre será preferible 1D por filas vs. 1D por columnas, y como 2D ya era peor, la elección para estos 3 supuestos es 1D por filas.
Ahora bien, la forma de agrupar óptima es otra: por diagonales; y más concretamente de forma cíclica, como se muestra en la figura adjunta suponiendo que se dispone de 3 procesadores. Aunque el balanceo de la carga puede no ser total, para valores de N y M grandes, respecto a p, su impacto se puede "despreciar" (aproximar).



El resto de cálculos son sencillos y equivalentes a los utilizados en el ejercicio de referencia, ya citado. Así, el tiempo de ejecución paralelo es, aproximadamente, el secuencial dividido por p al no haber comunicaciones, la eficiencia es 1, el coste es $pT(NM)$, la función de sobrecarga es 0 y es escalable porque el algoritmo es óptimo (eficiencia 1 constate o, en otras palabras, la función de sobrecarga es 0).

Problema 21 (Curso académico 20/21, octubre 2020)

Sea el algoritmo adjunto y p ordenadores, cada uno con un procesador de un solo núcleo. Las constantes de comunicaciones son t_S y t_W y t_c la de cálculo; n divisible por p y $(p < n)$. Se han diseñado 2 algoritmos paralelos. El primero reparte los datos por columnas: cada ordenador tiene columnas completas y consecutivas de la matriz. El segundo por filas: cada ordenador tiene filas completas y consecutivas de la matriz. Cada equipo hace: a) Todos los cálculos posibles con sus datos, siendo el resultado un vector; b) Si el vector resultante en a) es una solución completa pasa a la fase c). En caso contrario, envía el vector obtenido en a) a un gestor que recolecta y fusiona los vectores de todos los ordenadores y, seguidamente, envía el resultado de la fusión (un vector) a todos ellos; c) Los ordenadores, con su solución completa de a) o con la recibida en b), completan sus cálculos. El gestor solo admite comunicaciones P2P y no puede ejecutar simultáneamente varias P2P. Se admiten aproximaciones como, por ejemplo, $n + 1 = n - 1 = n$.

Algoritmo Secuencial($M \in \mathbb{R}^{nxn}$) **Para j=1 hasta n** **suma=0** **Para i=1 hasta n** **suma = suma + M[i][j]** **Fin Para** **Para i=1 hasta n** **M[i][j] = M[i][j] / suma** **Fin Para** **Fin Para** **Fin Algoritmo**	Se pide: 1) De forma concisa y clara ¿Qué hace el algoritmo secuencial? **(0.25 puntos)** 2) Expresión del tiempo de ejecución secuencial **(0.5 puntos)** 3) Expresión del tiempo de ejecución paralelo por columnas **(0.75 puntos)** 4) Expresión del tiempo de ejecución por filas **(1 punto)** 5) $E(n,p)$ de los 2 algoritmos paralelos **(0.75 puntos)** 6) ¿Cuál es el mejor diseño paralelo?, razonar la respuesta en base a la $E(n,p)$ **(1 punto)** 7) $T_0(n,p)$ y la escalabilidad del mejor diseño paralelo **(0.75 puntos)**

Solución

1) Normaliza los elementos de la matriz por columnas, esto es, divide los elementos de la matriz por la suma de los elementos de su columna.
2) $T(n) = n(1 + n + n)t_c \cong n(n + n)t_c = 2n^2 t_c$
3) $T(n,p)_1 = \frac{n}{p}(1 + n + n)t_c \cong \frac{n}{p}(n + n)t_c = \frac{2n^2 t_c}{p}$. Nótese que en este diseño el ordenado tiene todos los elementos de cada columna que le corresponde procesar, es decir, pasa directamente de a) a c). Cada procesador procesa $\frac{n}{p}$ columnas.
4) Tal como está formulado el problema (forma de trabajar descrita en a), b) y c)) sería $T(n,p)_2 = \left(\frac{n}{p}(1 + n)t_c\right) + (t(b)) + \left(\frac{n}{p}nt_c\right)$, donde $\left(\frac{n}{p}(1 + n)t_c\right)$ corresponde a los cálculos que se pueden hacer en a), $(t(b))$ el coste de b) y $\left(\frac{n}{p}nt_c\right)$ los cálculos restantes (c)). Aproximando se tiene que $T(n,p)_2 = \frac{2n^2}{p}t_c + t(b)$. Respecto a $t(b)$, según se describe en el enunciado hay que 1) recolectar los p vectores de tamaño n con las sumas parciales de todas las columnas de sus filas, con comunicaciones P2P no solapadas, 2) hacer la suma de p vectores de tamaño n y 3) distribuir un vector de tamaño n a los p ordenadores con comunicaciones P2P no solapadas. Por tanto: $p(t_s + nt_w) + pnt_c + p(t_s + nt_w)$. Entonces, $T(n,p)_2 = \frac{2n^2}{p}t_c + p(t_s + nt_w) + pnt_c + p(t_s + nt_w) = \left(\frac{2n^2}{p} + pn\right)t_c + 2p(t_s + nt_w)$.
5) $E(n,p)_1 = \frac{2n^2 t_c}{p\left(\frac{2n^2 t_c}{p}\right)}$=1. $E(n,p)_2 = \frac{2n^2 t_c}{p\left(\left(\frac{2n^2}{p}+pn\right)t_c+2p(t_s+nt_w)\right)}$
6) La eficiencia del primer diseño es máxima y constante $(E(n,p)_1 = 1)$. La del segundo es menor que 1 y decreciente al aumentar el valor de p. En consecuencia, el mejor diseño es "*por columnas*".
7) $T_0(n,p) = pT(n,p) - T(n) = p\left(\frac{2n^2 t_c}{p}\right) - 2n^2 t_c = 0$. No hay sobrecarga, por lo que el algoritmo es escalable. Esto ya se sabía porque su eficiencia es 1 y constante o, lo que es lo mismo, es de coste óptimo al ser $C(n,p) = T(n)$.

Preguntas que no se ha formulado y podrían haberse preguntado:

1) En el peor diseño paralelo ¿si se cumple que $t_s = t_w = t_c = 1$ y se usan las aproximaciones permitidas cómo es $E(n,p)$?

$$E(n,p)_2 = \frac{2n^2}{p\left(\left(\frac{2n^2}{p} + pn\right) + 2p(1+n)\right)} \cong \frac{2n^2}{2n^2 + np^2 + 2np^2} \cong \frac{2n^2}{2n(n+p^2)} = \frac{n}{n+p^2} = \frac{1}{1+\frac{p^2}{n}}$$

2) $T_0(n,p)$.

$$T_0(n,p) = p\left(\frac{2n^2}{p} + pn\right)t_c + 2p(t_s + nt_w) - 2n^2 t_c = np^2 t_c + 2p(t_s + nt_w)$$

3) Escalabilidad del peor diseño paralelo.

- Por el método "estándar"

$$t_c: n^2 = W \propto Knp^2 = KW^{\frac{1}{2}}p^2; W^{\frac{1}{2}} \propto Kp^2; W \propto (Kp^2)^2 \in O(p^4) \rightarrow No\ escalable$$
$$t_s: W \propto Kp \in O(p)$$
$$t_w: W \propto Knp = KW^{\frac{1}{2}}p; W^{\frac{1}{2}} \propto Kp; W \propto (Kp)^2 \in O(p^2)$$

- Por la comparación de órdenes se tiene $I(W,p) = \frac{T_0(n,p)}{T(n)} = \frac{O(np^2)}{O(n^2)}$, donde el orden del numerador no es igual o inferior al del denominador. Por tanto, no es escalable.

Problema 22 (Curso académico 20/21, diciembre 2020)

Sea $A \in \mathcal{R}^{nxm}$ una matriz y p el número de ordenadores conectados por una red de datos, siendo t_s la constante de establecimiento y t_w la inversa del ancho de banda. Cada ordenador tiene un procesador con un solo núcleo (*core*), siendo t_c la constante de cálculo, y 0 el coste de las operaciones de sincronicación/comunicación internas.

El problema a resolver consiste en aplicar repetidas veces un proceso. En cada repetición hay que, entre otras operaciones, "realizar una normalización por filas de todos los elementos de A".

Se pide:

1. **Aplicar las 2 primeras etapas de la Metodología de Diseño** (tipo de descomposición y análisis de las comunicaciones) para tener *el mejor* modelo (grafo). Sea metódico y exhaustivo.
2. **Aplicar la 3ª etapa de la Metodología de Diseño (agrupar)** seleccionando uno de los mejores particionados posibles. Justificar la elección con un estudio teórico completo.
3. **En 1. y en 2.** obtener las ecuaciones del tiempo secuencial, del tiempo paralelo, de la eficiencia, del coste y de la función de sobrecarga. En **2.**, además, realizar el estudio de la escalabilidad.

Solución

No se dice en el enunciado el tipo de normalización: en el intervalo $[0, 1]$, en el intervalo $[a, b]$, dividir por el máximo, etc. En todos los casos se debe calcular primero algún estadístico (el máximo, mínimo, suma, etc.), es decir, aplicar una *operación de reducción* en terminología de PCP. Y, en segundo lugar, actualizar todos los valores en función del tipo de normalización. Como no se indica se supondrá una normalización básica y sencilla.

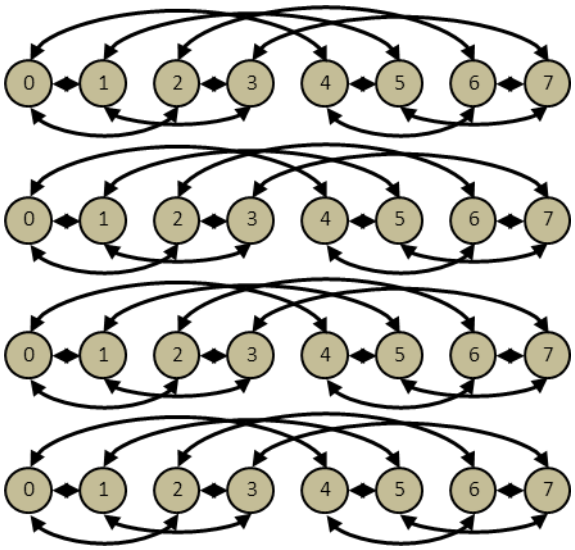
1. Aplicar las 2 primeras etapas de la Metodología de Diseño

Se opta por una descomposición del domino dado que "*todos y cada uno de los elementos de la matriz*" se deben actualizar, de forma iterativa, por el valor obtenido con un cálculo sencillo. Como solo hay una estructura, la matriz, sus elementos serán los nodos del grafo a construir. Inicialmente tantos nodos (tareas o procesadores) como elementos tenga la matriz.

Para el estudio de las comunicaciones (aristas o arcos del grafo) se debe recordar:

1) El problema de los reductores ha sido ampliamente estudiado en clase. El primer diseño, el más simple, es "*un procesador recibe todos los datos, calcula el reductor y, finalmente, envía al resto el estadístico calculado (los estadísticos calculados)*". Es decir, comunicaciones globales de $m\colon 1$ y luego de $1\colon m$. Cabe recordar que esta operación se hace a nivel de fila (o columna según la modalidad de examen) y que se ejecuta de forma independiente y concurrente sobre todas las filas. El dibujo resultante se puede consultar en la transparencia 60 del tema "Fundamentos del Diseño de Algoritmos Paralelos". Para este diseño se acepta (en PCP) que las comunicaciones $m\colon 1$ (o $1\colon m$) se modelizan con m operaciones P2P, es decir, el tiempo total de las comunicaciones a nivel de fila sería $2m(t_s + t_w)$.
   Por tanto, el tiempo total del algoritmo sería el tiempo empleado a nivel de fila. Y este es: $T(n, m, p) = (m-1)(t_s + t_w + t_c) + (m-1)(t_s + t_w) + t_c$. El primer término $\big((m-1)(t_s + t_w + t_c)\big)$ corresponde con la recepción (P2P) en una tarea de todos y cada uno de los datos de las otras tareas $\big((m-1)(t_s + t_w)\big)$ con las que comparte fila y la actualización del estadístico que se calcule, que se supone de coste 1 flop $\big((m-1)t_c\big)$. El segundo $\big((m-1)(t_s + t_w)\big)$ es el envío (P2P) del estadístico desde una tarea al resto y, el tercero y último (t_c), la actualización del valor de la tarea.
2) En esa misma transparencia se indica que las comunicaciones globales no son eficientes y que se deben, siempre que sea posible, transformar en locales. Así, para el problema de los reductores se explica que aplicando la técnica de "*Divide y Vencerás*" se puede hacer la transformación (ver dibujos en las transparencias 60 y 61). El resultado es que el coste de las comunicaciones a nivel de fila pasa a ser $2\log_2 m\,(t_s + t_w)$. El tiempo total del algoritmo se obtendría de forma similar a 1).
3) Ya en la transparencia 76, del mismo tema, se explica cómo la técnica "*Replicar*" es útil, en este problema, para que todos los procesadores (o tareas, por filas o columnas según el caso) obtengan el valor del reductor en el mismo tiempo que tarda la técnica de "*Divide y Vencerás*" estándar, resultando un tiempo de comunicaciones $\log_2 m\,(t_s + t_w)$.

En resumen, para una matriz de $8x4$, se obtiene el siguiente grafo.



Entre filas los cálculos son independientes y totalmente concurrentes. Dentro de cada fila, el tiempo necesario vendrá determinado por las comunicaciones y la actualización que se supondrá de coste $1\ flop$.

Por todo ello:

$$T(n,m) = 2nmt_c$$

$$T(n,m,p) = \log_2 m\,(t_s + t_w + t_c) + t_c$$

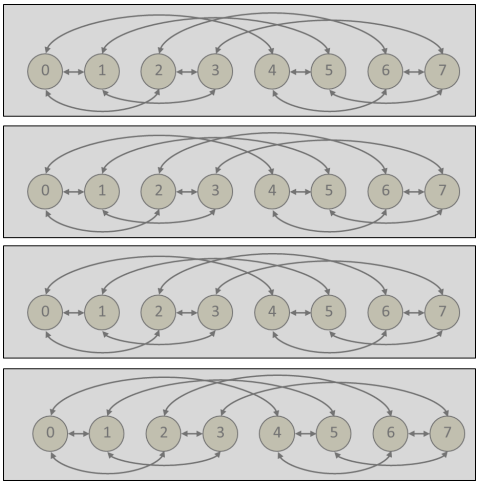
$$E(n,m,p) = \frac{2nmt_c}{nm(\log_2 m\,(t_s + t_w + t_c) + t_c)} = \frac{2t_c}{\log_2 m\,(t_s + t_w + t_c) + t_c} \cong \frac{1}{\log_2 m}$$

Nótese que en el diseño $p = nm$. La eficiencia es monótona decreciente con valores bajos (p. ej. para $m = 1024$ la eficiencia sería 0.1).

Claves de la solución de este apartado: Descomposición del Dominio, Divide y Vencerás, Réplica de comunicaciones/cálculo.

2. Aplicar la 3ª etapa de la Metodología de Diseño

Hay sobrados ejemplos en los apuntes similares a este. Es decir, ejemplos donde un tipo de agrupamiento anula todas las comunicaciones, resultado una relación superficie-volumen óptima, de valor 0. En este caso es el agrupamiento óptimo es por "bloques de filas", consecutivas o no (cíclico), resultado el dibujo adjunto.



Por tanto:

$$T(n,m) = 2nmt_c$$

$$T(n,m,p) = \frac{2nmt_c}{p}$$

$$E(n,m,p) = \frac{2nmt_c}{p\frac{2nmt_c}{p}} = 1$$

$$C(n,m,p) = pT(n,m,p) = p\frac{2nmt_c}{p} = 2nmt_c$$

$$T_0(n,m,p) = C(n,m,p) - T(n,m) = 2nmt_c - 2nmt_c = 0$$

En resumen, eficiencia máxima y constante, algoritmo de coste óptimo, función de sobre carga igual a 0. Consecuentemente, el algoritmo es escalable, no es preciso realizar cálculo alguno.

Claves de la solución de este apartado: Agrupar por bloques de filas, relación superficie/volumen con valor 0, valores óptimos de los parámetros relativos, escalable.

NOTA. La solución para el *Modelo A* es similar. A nivel de ecuaciones la única diferencia está en el modelado de las comunicaciones, que pasa de ser $\log_2 m\,(t_s + t_w)$ a ser $\log_2 n\,(t_s + t_w)$.

Problema 23 (Curso académico 21/22, noviembre 2021)

Sea $A \in \mathcal{R}^{nxn}$ una matriz donde sus elementos deben ser actualizados con el valor de un estadístico sencillo (máximo, media, etc.) que se calcular usando todos los elementos de A. Por simplicidad, se supone que "el coste de actualizar el estadístico (por cada elemento de A) y cada elemento de A (con el estadístico) es un FLOP". Se usa descomposición del domino. Se pide:

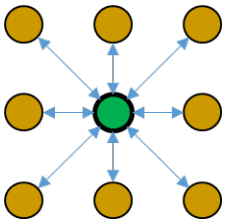
1. Ecuación del $T(n)$ **(0.5 puntos)**
2. Análisis de las comunicaciones, 2ª etapa de la MDAP, construyendo el grafo y presentando las ecuaciones del $T(n,p)$, de la $E(n,p)$ y de la $T_O(n,p)$ para:
   2.1. Un diseño paralelo con cota de complejidad mayor o igual a la de $T(n)$ **(1.0 puntos)**
   2.2. Un diseño paralelo con cota de complejidad inferior pero influenciada por n^2 **(1.0 puntos)**
   2.3. Un diseño paralelo con cota de complejidad inferior pero influenciada por n **(2.5 puntos)**

Se admite la aproximación $x - 1 = x$ y similares. Dibuje los grafos suponiendo $n = 3$ para el primer diseño, $n = 2$ para el segundo y $n = 3$ para el tercer diseño. Identifique, si es el caso, las técnicas usadas para transformar un grafo en otro.

Solución

Apartado 1. Siguiendo las indicaciones del enunciado, 1 FLOP por operación, $T(n) = 2n^2 t_c$

Apartado 2.1. Usando comunicaciones globales para calcular el estadístico y su distribución con $n = 3$ un grafo resultante sería el que se muestra en la siguiente figura.



La ecuación del tiempo de ejecución paralelo sería

$$T(n,p) = (n^2 - 1)(t_s + t_w) + (n^2 - 1)t_c + (n^2 - 1)(t_s + t_w) + t_c,$$

donde el primer $(n^2 - 1)(t_s + t_w)$ denota el tiempo para que el nodo (procesador) elegido (vale cualquiera) reciba todos los datos, $(n^2 - 1)t_c$ el coste de computar el estadístico, el segundo $(n^2 - 1)(t_s + t_w)$ corresponden con la distribución del estadístico calculado y t_c la actualización concurrente de los n^2 elementos de la matriz. Utilizando las aproximaciones permitidas se tiene

$$T(n,p) \cong (2(t_s + t_w) + t_c)n^2$$

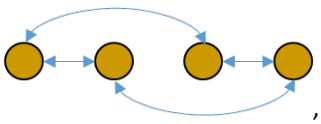
$$E(n,p) \cong \frac{2n^2 t_c}{p(2(t_s + t_w) + t_c)n^2} = \frac{2t_c}{p(2(t_s + t_w) + t_c)}$$

Como en esta etapa del diseño $p = n^2$ entonces

$$E(n,p) \cong \frac{2t_c}{p(2(t_s + t_w) + t_c)} = \frac{2t_c}{n^2(2(t_s + t_w) + t_c)} \cong \frac{t_c}{(t_s + t_w + t_c)n^2}$$

$$T_o(n,p) \cong pT(n,p) - T(n) = p(2(t_s + t_w) + t_c)n^2 - 2n^2 t_c = 2n^4(t_s + t_w) + (n^4 - 2n^2)t_c$$

Apartado 2.2. Aplicando la técnica de divide y vencerás y replicando comunicaciones (ver transparencias del tema de teoría relacionado y los múltiples ejemplos en el boletín de problemas de examen resueltos) para n =2 el grafo se transformaría en el que se muestra en la siguiente figura



,

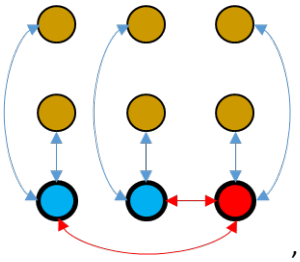
que se sabe corresponde a una reducción logarítmica con lo que las ecuaciones resultantes son

$$T(n,p) \cong (2(t_s + t_w) + t_c)\log_2 n^2$$

$$E(n,p) \cong \frac{2n^2 t_c}{p(2(t_s + t_w) + t_c)\log_2 n^2} = \frac{2n^2 t_c}{n^2(2(t_s + t_w) + t_c)\log_2 n^2} \cong \frac{t_c}{(t_s + t_w + t_c)\log_2 n^2}$$

$$T_o(n,p) \cong p(2(t_s + t_w) + t_c)\log_2 n^2 - 2n^2 t_c = 2n^2 \log_2 n^2\,(t_s + t_w) + (n^2 \log_2 n^2 - 2n^2)t_c$$

Apartado 2.3. La idea subyacente es aplicar las técnicas usadas en 2.2 en dos pasos para aprovechar al máximo la potencial concurrencia entre los distintos procesadores al tratarse de un estadístico simple, donde no importa el orden al procesar los elementos. Así, por ejemplo, se resolvería el subproblema a nivel de columnas y, con posterioridad, el problema general, pero sobre un vector (n) y no la matriz completa (n^2). Gráficamente



,

donde el color azul de los arcos denota la primera fase, operaciones solapadas por columnas, y el color rojo la segunda fase. Obviamente, hay una tercera fase idéntica a la primera a nivel gráfico que corresponde con la recepción del estadístico calculado. La ecuación del tiempo de ejecución paralelo sería

$$T(n,p) = (t_s + t_w + t_c)\log_2 n + (t_s + t_w + t_c)\log_2 n + (t_s + t_w)\log_2 n + t_c,$$

donde el primer $(t_s + t_w + t_c)\log_2 n$ es el tiempo por columnas, el segundo $(t_s + t_w + t_c)\log_2 n$ es el tiempo sobre el vector (rojo en la figura), $(t_s + t_w)\log_2 n$ es la distribución del estadístico por columnas y el t_c final la actualización concurrente de todos los elementos de la matriz. Simplificando o aproximando:

$$T(n,p) \cong (t_s + t_w + t_c)3\log_2 n$$

$$E(n,p) \cong \frac{2n^2 t_c}{p(t_s + t_w + t_c)3\log_2 n} = \frac{2n^2 t_c}{n^2(2(t_s + t_w) + t_c)3\log_2 n} \cong \frac{t_c}{(t_s + t_w + t_c)\log_2 n}$$

$$T_o(n,p) \cong p(t_s + t_w + t_c)3\log_2 n - 2n^2 t_c = 3n^2 \log_2 n\,(t_s + t_w) + (3n^2 \log_2 n - 2n^2)t_c$$

Comparando las eficiencias

$$E(n,p) \cong \frac{t_c}{(t_s + t_w + t_c)n^2} \cong \frac{1}{n^2} \qquad 2.1$$

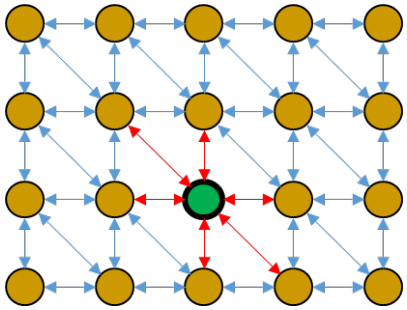
$$E(n,p) \cong \frac{t_c}{(t_s + t_w + t_c)\log_2 n^2} \cong \frac{1}{\log_2 n^2} \qquad 2.2$$

$$E(n,p) \cong \frac{t_c}{(t_s + t_w + t_c)\log_2 n} \cong \frac{1}{\log_2 n} \qquad 2.3$$

Que corresponde con las expectativas del enunciado.

Problema 24 (Curso académico 21/22, noviembre 2021)

Sea $A \in \mathcal{R}^{mxn}$ una matriz, con $n > m$, y el grafo adjunto, para $m = 4$ y $n = 5$, de un diseño realizado para un proceso que consiste en "… actualizar todos los elementos de la matriz …", de coste un FLOP por elemento. Se necesita saber qué agrupamiento de los seleccionados es el mejor para un sistema de memoria distribuida con p procesadores y constantes t_c, t_s y t_w, suponiendo que m y n son divisibles por p y que el coste de las comunicaciones de la distribución/recolección de la matriz inicial/final es 0.



Los agrupamientos seleccionados son: a) 1D por Bloques de Filas Consecutivas (BFC) y b) 1D por Bloques de Columnas Consecutivas (BCC). Se pide:

1. Ecuación del de ejecución secuencial **(0.5 puntos)**
2. Ecuación del tiempo de ejecución paralelo para 1D BFC **(1.0 puntos)**
3. Ecuación del tiempo de ejecución paralelo para 1D BCC **(1.0 puntos)**
4. Relación superficie volumen de los 2 agrupamientos y decisión **(1.0 puntos)**
5. ¿es escalable el diseño seleccionado? **(1.5 puntos)**

Solución

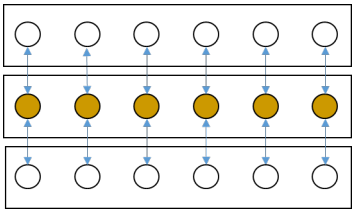
A lo que el alumno debe responder en este problema ya está resuelto en varios ejemplos del tema de teoría relacionado, así como en múltiples ejercicios del boletín de problemas de examen resueltos. Nótese que la parte diferenciadora, diseño inicial, y aspectos potencialmente difíciles, como agrupamientos cíclicos y/o por bloques 2D y escalabilidad sobre ecuaciones complejas, no son objetivo de las preguntas formuladas.

Por otra parte, como se ha comentado, lo relevante al agrupar (y en la relación superficie-volumen) no es "el uso" de la información, sino su necesidad (si hay que intercambiarla o no).

Por ejemplo, para un agrupamiento 1D BCC donde la columna *j-esima* es la última del procesador k y, por tanto, la columna *(j+1)-esima* es la primera del procesador *k+1*, el elemento $A[i,j]$, que tiene el procesador k, lo necesita el procesador *k+1* para modificar son elementos $A[i,j+1]$ y $A[i+1,j+1]$. Esto implica que el elemento $A[i,j]$ debe ser enviado del procesador k al *k+1*, pero una sola vez (no hay que enviarlo 2 veces porque el procesador *k+1* lo use para modificar dos de sus elementos).

Apartado 1. Siguiendo las indicaciones del enunciado, 1 FLOP por operación, $T(m,n) = mnt_c$

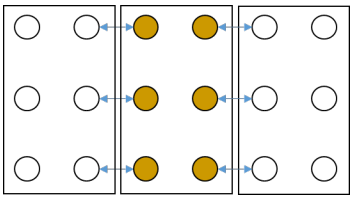
Apartado 2. El siguiente dibujo muestra el resultado suponiendo p=3 y resaltando las comunicaciones sobre el procesador del medio.



$$T(m,n,p) \cong \frac{mnt_c}{p} + 2(t_s + nt_w)$$

Se ha supuesto, sin pérdida de generalidad, que el intercambio de información P2P se puede hacer con una sola operación de mensajería.

Apartado 3. Procediendo de igual forma se tiene la siguiente figura y ecuación del tiempo de ejecución paralelo.



$$T(m,n,p) \cong \frac{mnt_c}{p} + 2(t_s + mt_w)$$

Apartado 4. La única diferencia entre los agrupamientos BFC y BCC es el coeficiente que acompaña a la constate t_w, que es la variable n en BFC y la m en BCC. Como $n > m$ entonces de los dos BCC es el mejor. Su relación superficie-volumen, es

$$\frac{2(t_s + mt_w)}{\frac{mnt_c}{p}} \cong \frac{2p(t_s + mt_w)}{mnt_c}$$

Apartado 5. Análisis de la escalabilidad.

$$T_o(m,n,p) \cong p\left(\frac{mnt_c}{p} + 2(t_s + mt_w)\right) - mnt_c = 2p(t_s + mt_w)$$

$$I_{t_s}: W \propto Kp \in O(p)$$

$$I_{t_w}: W \propto Kmp$$

W es mn. Al tratarse de una matriz no cuadrada el incremento de W puede hacerse sobre m, sobre n o sobre ambos a la vez. El incremento sobre ambos a la vez, dado que I_{t_w} no depende de n, es equivalente o similar al incremento sobre m. Esta situación "particular" se ha tratado/discutido un ejemplo de las transparencias del tema de teoría relacionado. En resumen, si se aborda por la relación de ordenes se tiene que

$$I_{t_w}(W,p) = \frac{O(mp)}{O(mn)} = \frac{O(p)}{O(n)},$$

concluyendo que el algoritmo es potencialmente escalable. Si se aborda por el método "clásico" y sabiendo que n no influye se tiene que

$$I_{t_w}: W \propto Kmp; n \propto Kp; n \propto Kp \in (p),$$

llegando a la misma conclusión.