Eficiencia y Aceleración

Robinson Rivas,
Fundamentos de
Programación Paralela

El problema base

- El paralelismo se basa en la ***coordinación*** de tareas para trabajar en un problema común
- Esta coordinación exige operaciones ***adicionales*** a las que tenía el problema original
- Dependiendo de la arquitectura, además hay un tiempo que llamamos ***overhead*** que se debe dedicar al procesamiento de datos, y un tiempo de ***comunicaciones*** que no existe en el problema original

Definiciones

- Supongamos que se tiene un programa secuencial que resuelve el problema A de forma eficiente
- ¿qué ocurre si ejecutamos ese programa sobre dos procesadores en lugar de uno solo?

Definiciones

- Para observar el comportamiento, note que no debemos ejecutar ***cualquier programa*** para obtener el tiempo secuencial. Debemos escoger ***el mejor*** y ejecutarlo sobre la misma plataforma paralela
- Esto nos da un tiempo estadísticamente medido, al que llamamos (sorprendentemente) ***Tiempo Secuencial Ts***

Definiciones

- Esto lleva a la pregunta natural: ¿qué tan rápido puede llegar a ser nuestro programa paralelo?
- Para estos ejemplos, supongamos que se cuenta con *p* procesadores homogéneos
- Si los procesadores ***no son homogéneos*** debe evaluarse el tiempo del más lento
  - (recuerde que una manada camina al ritmo del más lento)

Definiciones

- Sea ***Ts*** el tiempo Secuencial.
  - Usualmente se ejecuta el mejor programa secuencial en un solo procesador
- Sea ***Tp*** el tiempo Paralelo para ***p*** procesadores
  - Se miden varias ejecuciones y se promedian

Aceleración (Speedup)

- La Aceleración (***Sp***) se mide con la fórmula

$$Sp = Ts/Tp$$

Observe que esperamos que el tiempo paralelo sea menor, por lo que la aceleración es un número > 1

Aceleración (speedup)

- La aceleración nos indica ***qué tan rápido es nuestro programa paralelo***

Pregunta: ¿Es bueno que un programa paralelo sea 10 veces más rápido que el secuencial?

Eficiencia (performance)

- La Eficiencia (***Ep***) se mide con la fórmula

$$Ep = Sp/p = Ts/pTp$$

En este caso, la Eficiencia usualmente será un número >0 y <1

NOTA: ¡¡en algunos casos muy especiales la eficiencia puede ser mayor a 1!!

Medidas

En condiciones ideales:

$$S_p \to p$$

$$E_p \to 1$$

En la práctica:

$$S_p \leq p$$

$$E_p \leq 1$$

Efectividad

- Podemos conocer el costo de nuestro programa calculando la Efectividad:

  Fp = Sp / pTp

  Luego si ***Cp = pTp*** → ***Fp = Sp / Cp***

- A ***Cp*** lo asociamos al costo de la solución. La Efectividad asocia la Aceleración y la Eficiencia

Efectividad

- Podemos ver esta medida como la suma total de operaciones que hacen los procesadores paralelos
- Dado que el Costo es ***pTp,*** la suma total de tiempos da una medida de lo costoso que resulta tener ***p*** procesadores
- En le mundo moderno esto se asocia a costos económicos y energéticos principalmente

Comunicaciones

- En presencia de comunicaciones, el diferencial de tiempo se debe normalmente al tiempo de comunicaciones más el overhead (tiempo de coordinación)
- Así, ***Tp = Tcom + Tcoo + Tcal***
- Donde:
  - ***Tcom = comunicaciones***
  - ***Tco = coordinación***
  - ***Tcal = cálculo o cómputo***

Revisando la Aceleración

- Speedup

$$S_p = \frac{t_s}{t_p} = \frac{t_s}{\frac{t_s}{p} + t_{com} + t_{coo}} = \frac{1}{\frac{1}{p} + \frac{t_{com}+t_{coo}}{t_s}} = \frac{p}{1 + p\frac{(t_{com}+t_{coo})}{t_s}} = \frac{p}{1 + \frac{(t_{com}+t_{coo})}{t_{cal}}}$$

- Eficiencia

$$E_p = \frac{S_p}{p} = \frac{1}{1 + \frac{t_{com}+t_{coo}}{t_{cal}}} = \frac{1}{1+\omega}$$

donde

$$\omega = \frac{t_{com} + t_{coo}}{t_{cal}}$$

refleja la sobrecarga -o pérdida de eficiencia- debido al paralelismo.

¿Todo es paralelizable?

- En un programa hay partes que NO pueden ser paralelizadas, debido a muchos factores
- Esto hace que los tiempos nunca converjan a los ideales
- La Ley de Amdahl estima este problema para programas genéricos (no solo paralelos)
- De acuerdo al principio de Amdahl, la ***fracción paralela*** determina el tope de eficiencia que se puede obtener de un programa

Amdahl

Se puede definir una fracción paralelizable del programa (f_p) y una serial (o secuencial) (f_s)

$$f_p + f_s = 1$$

El tiempo de procesamiento secuencial:

$$t_s = (1 - f_p)\, t_s + f_p\, t_s$$

y el de procesamiento paralelo con p procesadores:

$$t_p = (1 - f_p)\, t_s + \frac{f_p}{p}\, t_s$$

Amdahl

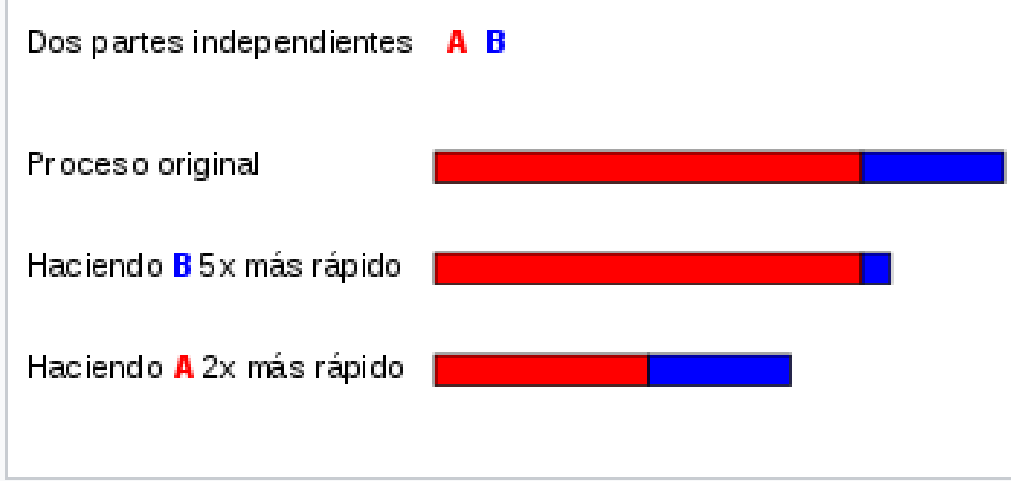
Speedup:

$$S_p = \frac{t_s}{t_p} = \frac{1}{(1 - f_p) + \frac{f_p}{p}} = \frac{p}{p\,(1 - f_p) + f_p}$$

Eficiencia:

$$\eta_f = E_p = \frac{S_p}{p} = \frac{1}{p\,(1 - f_p) + f_p}$$

Ejemplo



Asumiendo que una tarea tiene dos partes independientes, **A** y **B**, consumiendo **B** el 25% del tiempo total de computación. Trabajando muy duro se puede realizar **B** 5 veces más rápido y sin embargo esto sólo reduce el tiempo de computación un poco; en contraste, una pequeña mejora de **A** hace que ésta vaya el doble de rápido. Esto hace que sea mucho mejor la optimización de **A** que de **B** aunque se mejore mucho más **B** (5x contra 2x).

Fuente: wikipedia

Amdahl

La **ley de Amdahl** es, en ciencia de la computación, formulada por Gene Amdahl, utilizada para averiguar la mejora máxima de un sistema de información cuando solo una parte de éste es mejorado.

Establece que:

> La mejora obtenida en el rendimiento de un sistema debido a la alteración de uno de sus componentes está limitada por la fracción de tiempo que se utiliza dicho componente.

La fórmula original de la ley de Amdahl es la siguiente:

$$T_m = T_a \cdot \left((1 - F_m) + \frac{F_m}{A_m} \right)$$

siendo:

- F_m = fracción de tiempo que el sistema utiliza el subsistema mejorado
- A_m = factor de mejora que se ha introducido en el subsistema mejorado.
- T_a = tiempo de ejecución antiguo.
- T_m = tiempo de ejecución mejorado.

Fuente: wikipedia

Amdahl

Esta fórmula se puede reescribir usando la definición del incremento de la velocidad que viene dado por $A=T_a/T_m$, por lo que la fórmula anterior se puede reescribir como:

$$A = \frac{1}{(1 - F_m) + \frac{F_m}{A_m}}$$

siendo:

- A es la aceleración o ganancia en velocidad conseguida en el sistema completo debido a la mejora de uno de sus subsistemas.
- A_m , es el factor de mejora que se ha introducido en el subsistema mejorado.
- F_m , es la fracción de tiempo que el sistema utiliza el subsistema mejorado.

Por ejemplo, si en un programa de ordenador el tiempo de ejecución de un cierto algoritmo supone un 30% del tiempo de ejecución total del programa, y conseguimos hacer que este algoritmo se ejecute en la mitad de tiempo se tendrá:

- $A_m = 2$
- $F_m = 0.3$
- $A \approx 1.18$

Fuente: wikipedia

Ejemplo 1

- Tome como ejemplo dos arreglos del cual quiere sumar el producto todos sus componentes
- ¡Increíblemente, este problema está en el corazón de la mayoría de los modelos físicos y matemáticos que resuelven problemas muy complejos!

Ejemplo 1

```
function prod(float A[],B[],int N)
{  float sum=0;
   int i;
   for (i=0..N-1)
        sum=sum+A[i]*B[i]
   return sum;
}
```

Ejemplo 1

- Observe que el programa secuencial es extraordinariamente simple conociendo el tamaño N de los dos arreglos
- Asuma:
  - La llamada a la función tarda 1 ut
  - La multiplicación flotante tarda 5 ut
  - La suma flotante tarda 3 ut
  - Las comunicaciones, de hacerse, tardan 6 ut
  - El ciclo, return y otras operaciones tardan 0 ut (despreciables en este contexto)

Ejercicio 1

1. Indique el costo en tiempo para entradas N=102,104,106,108
2. Proponga un algoritmo paralelo
3. Calcule para ese algoritmo el tiempo para N= 102,104,106,108 usando 2,4,8 y 100 procesadores
4. Discuta los resultados

Ejemplo 2

- Tome como ejemplo la búsqueda de un elemento en un arreglo desordenado
- ¡Increíblemente, este problema TAMBIÉN está en el corazón de la mayoría de los modelos físicos y matemáticos que resuelven problemas muy complejos!

Ejemplo 2

```
function search(float A[],element, int N)
{ int i=0;
  boolean found=false;
  while (i<N and not found)
        { if A[i]=element then found=true;
          else i=i+1;
        }
  return i
}
// retorna la posición, o N si no está
```

Ejemplo 2

- Observe que el programa secuencial es extraordinariamente simple conociendo el tamaño N del arreglo
- Asuma:
  - La llamada a la función tarda 1 ut
  - La suma entera tarda 3 ut
  - Las comunicaciones, de hacerse, tardan 6 ut
  - Las comparacions, return y otras operaciones tardan 0 ut (despreciables en este contexto)

Ejercicio

1. Indique el costo ***promedio*** en tiempo para entradas N=102,104,106,108
2. Proponga un algoritmo paralelo
3. Calcule para ese algoritmo el tiempo para N= 102,104,106,108 usando 2,4 y 8 procesadores en estos escenarios:
    1. El elemento no está en el 95% de los casos
    2. El elemento está en la posición N/2 + 1 en el 95% de los casos
    3. El elemento está en la posición N/3 + 1 en el 45%de los casos y en la posición N/5+ 1 en el 45% de los casos
4. Discuta los resultados

Fuentes

- Victorio Sonzogni
  - http://venus.santafe-conicet.gov.ar/cursos/moodledata/17/Transp_eficiencia.pdf
- Wikipedia
  - https://es.wikipedia.org/wiki/Ley_de_Amdahl
- Ian Foster, design of parallel algorithms
  - https://www.mcs.anl.gov/~itf/dbpp/text/book.html