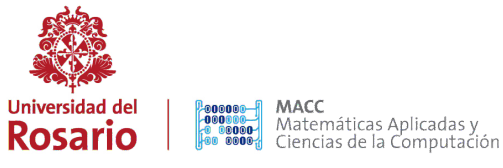
Modelo de computación

Carlos E. Alvarez[1].

[1]Dep. de Matemáticas aplicadas y Ciencias de la Computación, Universidad del Rosario

2020-I



Programación paralela

Objetivo: Disminuir el tiempo de ejecución de una tarea.

Programa dividido en parte no paralelizable (secuencial) y parte paralelizable

Algoritmo serial

$$T_1 = T_{sec} + T_{par} \tag{1}$$

Crear hilos de ejecución tiene un costo (overhead):

Algoritmo paralelo

$$T_P = T_{sec} + \left(T_{oh} + \frac{T_{par}}{P} \right) \tag{2}$$

En donde P es el número de unidades de ejecución.

Programación secuencial

Modelo RAM:

- Ejecución secuencial de instrucciones
- Asignación y operaciones básicas se realizan en una unidad de tiempo
- Unidades de información (tipos de datos primitivos) ocupan una unidad de memoria
- La memoria es infinita

Programación paralela

Expandir el modelo RAM:

- Capacidad de crear hilos de ejecución (**spawn**)
- Capacidad de sincronizar hilos de ejecución (**sync**)
- Capacidad de repartir las iteraciones de un ciclo en varios hilos (**parallel**)

Suma de vectores

Algoritmo serial:

```
1    V-Sum(x,y,n):
2      res is a new vector of length n
3      for i = 1 to n
4        res_i = x_i + y_i
5      return res
```

Ej: Implemente el algoritmo que suma vectores, inicializando los dos sumandos en 1 y 2 (El vector resultante debe tener 3 en todos sus elementos). Separe memoria del heap para guardar los vectores sumandos y el vector respuesta, usando `new`. No olvide liberar la memoria usando `delete[]`.

Suma de vectores

Algoritmo paralelo:

```
1   V-Sum(x,y,n):
2     res is a new vector of length n
3     parallel for i = 1 to n
4       res_i = x_i + y_i
5     return res
```

Algoritmo paralelo, implementando el **parallel for** con cuatro hilos:

```
1   V-Sum-Section(x,y,res,a,b)
2     for i = a to b
3       res_i = x_i + y_i
4
5   V-Sum(x,y,n):
6     res is a new vector of length n
7     chunk =  n / 4
8     spawn V-Sum-Section(x,y,res,chunk+1,2chunk)
9     spawn V-Sum-Section(x,y,res,2chunk+1,3chunk)
10    spawn V-Sum-Section(x,y,res,3chunk+1,4chunk)
11    V-Sum-Section(x,y,res,1,chunk)
12    sync
13    return res
```

Generando hilos con C++

Ejemplo

```
1   void func(int x, int& y) {
2     y = 2*x;
3   }
4
5   int main() {
6     int x1=1, x2=2;
7     int y1, y2;
8     thread thr1(func,x1,ref(y1));
9     thread thr2(func,x2,ref(y2));
10
11    thr1.join();
12    thr2.join();
13  }
```

Ej: Implemente el algoritmo paralelo con cuatro hilos para sumar vectores usando la librería `thread`. Asegúrese de la correctitud del resultado. Use la función

```
1  #include <sys/time.h>
2
3  double gettime() {
4    struct timeval tp;
5    gettimeofday(&tp, nullptr);
6    return tp.tv_sec + tp.tv_usec/(double)1.0e6;
7  }
```

para comparar los tiempos de ejecución de las versiones serial y paralela para diferentes tamaños del problema.

Secuencia de Fibonacci

$$F_n = F_{n-1} + F_{n-2}. \tag{3}$$

```
1  Fib(n)
2    if n ≤ 1
3      return n
4    else
5      x = Fib(n-1)
6      y = Fib(n-2)
7      return x + y
```

Tenemos que

$$T(n) \leq aF_n - b, \quad a > 1, b > 0. \tag{4}$$

Dem. (Inducción). Caso base:

$$T(1) \leq aF_1 - b = a - b \tag{5}$$
$$\Rightarrow 1 \leq a - b. \tag{6}$$

Suponga:

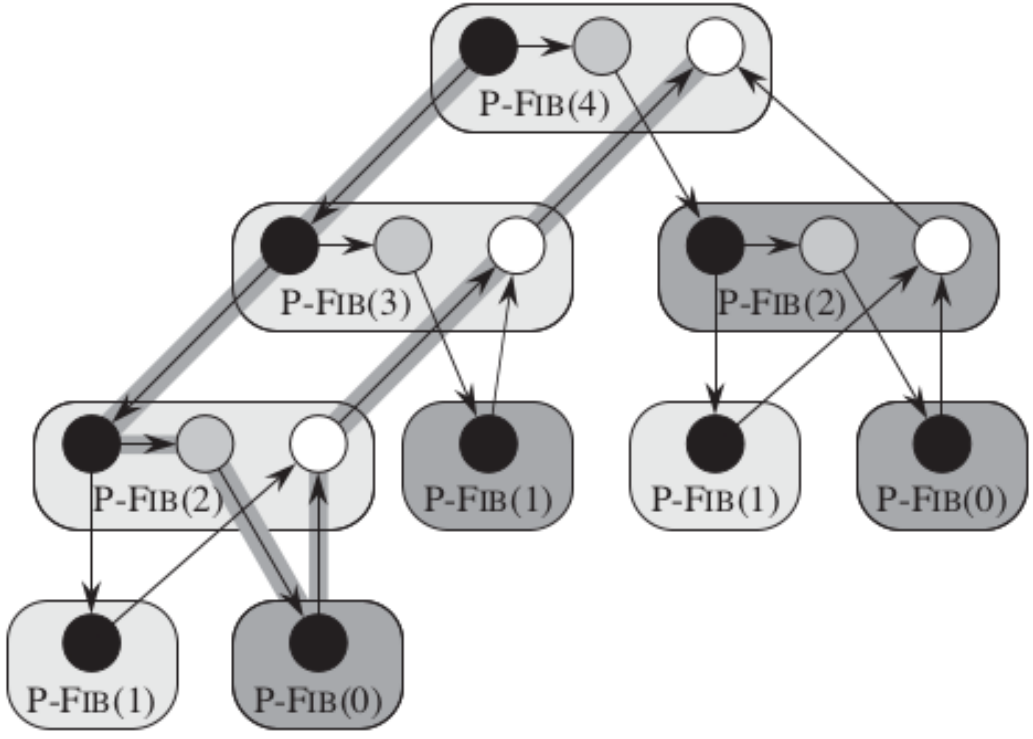
$$T(n-1) \leq aF_{n-1} - b \tag{7}$$
$$T(n-2) \leq aF_{n-2} - b \tag{8}$$

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + \Theta(1) \\ &\leq a(F_{n-1} + F_{n-2}) - 2b + \Theta(1) \\ &= aF_n - b + (\Theta(1) - b) \\ &\leq aF_n - b, \quad b \geq \Theta(1). \end{aligned} \tag{9}$$

Fibonacci con computación multi-hilo

```
1  P-Fib(n)
2    if n ≤ 1
3      return n
4    else
5      x = spawn P-Fib(n-1)
6      y = P-Fib(n-2)
7      sync
8      return x + y
```

Hebra*(strand)*: Grupo de instrucciones no paralelas (vértice).
Trabajo*(work)*: Número total de hebras (Cada operación implica un trabajo de 1).
Envergadura*(span)*: Longitud del mayor camino.



Medidas de rendimiento

$\boldsymbol{P}$: Número de procesadores.
$\boldsymbol{T_P}$: Tiempo en el que se ejecuta el proceso usando P procesadores.
$\boldsymbol{T_\infty}$: Tiempo en el que se ejecuta el proceso cuando se tienen suficientes procesadores para correr cada sub-proceso que se engendre. Nunca es menor que la envergadura.

- **Ley del trabajo:** Un proceso con P procesadores hace, a lo más, un trabajo total PT_P, que tiene como cota mínima el trabajo que se hace con un solo procesador: $1T_1$. Por lo tanto $T_P \geq T_1/P$.
- **Ley de la envergadura:** Un computador con P procesadores no puede ejecutar en menos tiempo que una máquina con procesadores ilimitados: $T_P \geq T_\infty$.
- **Aceleración (*speedup*):** $S(P) = T_1/T_P$. Si $S(P) = \Theta(P)$ se dice que la aceleración es lineal.
- **Paralelismo:** T_1/T_∞. Ejemplo: Si en el cálculo de `P-Fib(4)` cada hebra se ejecuta en tiempo unitario: $T_1/T_\infty = 17/8 = 2.125$.
- **Eficiencia:** $T_1/(PT_P)$.

Análisis del algoritmo de Fibonacci

Fibonacci secuencial: Los elementos de la secuencia pueden escribirse como

$$F_n = \left\lfloor \frac{\phi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor, \quad \phi = \frac{1+\sqrt{5}}{2}, \tag{10}$$

y por la ec. (9), tenemos para el algoritmo secuencial

$$T_1(n) = O(\phi^n). \tag{11}$$

Fibonacci concurrente: Nos interesa calcular la envergadura T_∞.

- La envergadura de dos sub-cómputos en serie es la suma de sus envergaduras
- La envergadura de dos sub-cómputos en paralelo es el máximo de las dos envergaduras

Para `P-Fib(n)` :

$$\begin{aligned} T_\infty(n) &= \max(T_\infty(n-1), T_\infty(n-2)) + \Theta(1) \\ &= T_\infty(n-1) + \Theta(1), \end{aligned} \tag{12}$$

cuya solución es lineal

$$T_\infty(n) = O(n). \tag{13}$$

El orden del paralelismo del algoritmo es entonces

$$T_1(n)/T_\infty(n) = O(\phi^n/n), \tag{14}$$

que crece con n.

Ej: Realice el análisis de los algoritmos serial y paralelo para la suma de vectores. Si el orden del tiempo de ejecución es el mismo, tenga en cuenta los factores que aparecen antes de tomarlo.

Argumento de Ahmdal

Defina la fracción serial:

$$\gamma = \frac{T_{sec}}{T_1} \tag{15}$$

Entonces

$$\begin{aligned} T_P &= T_{sec} + \frac{1}{P}(T_1 - T_{sec}) + T_{oh} \\ &= \gamma T_1 + \frac{1}{P}(T_1 - \gamma T_1) + T_{oh} \\ &= T_1 \left(\gamma + \frac{1}{P}(1 - \gamma) + \gamma_{oh} \right), \end{aligned} \tag{16}$$

en donde $\gamma_{oh} = T_{oh}/T_1$.

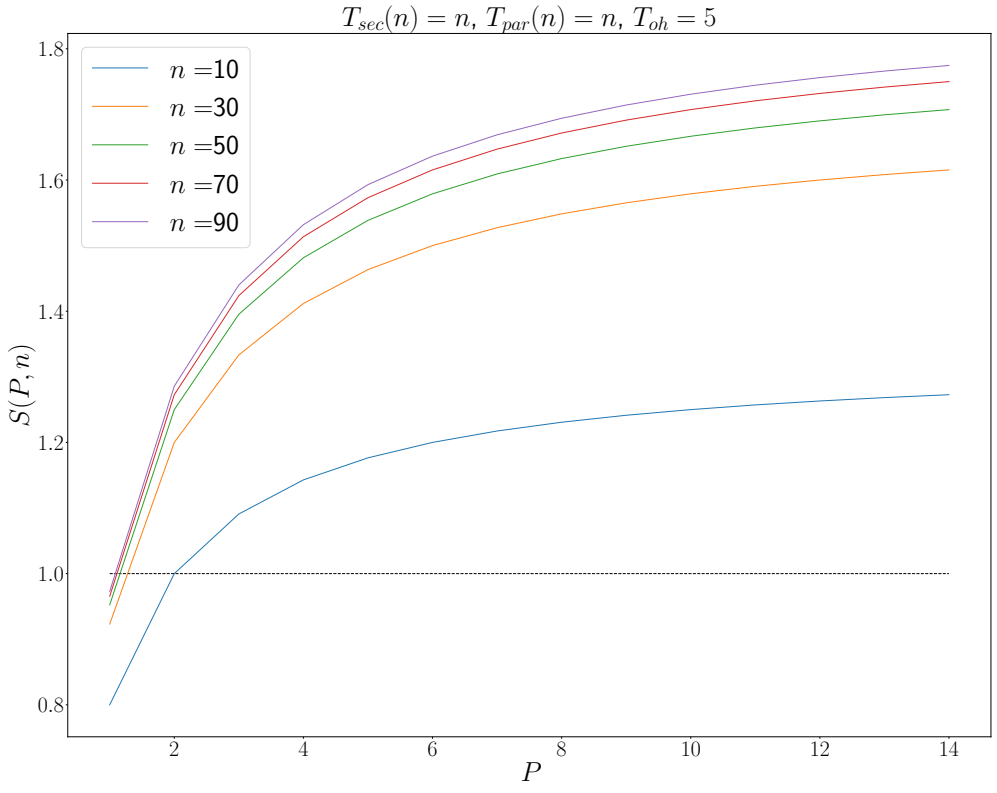
Por lo tanto:

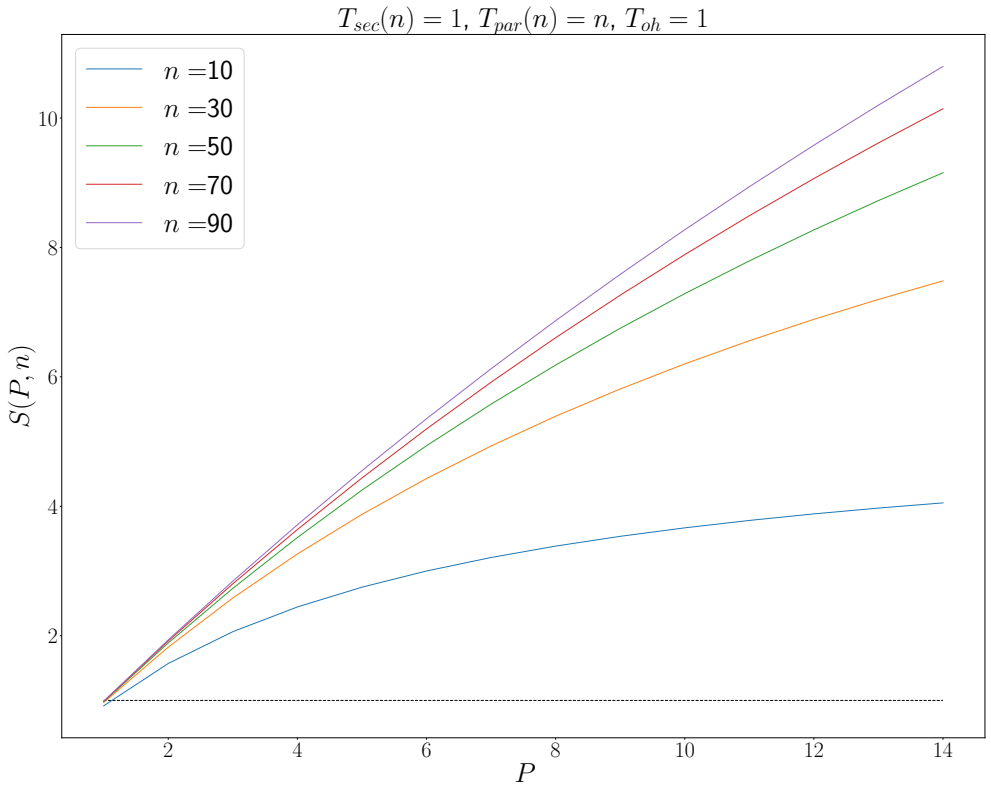
$$S(P) = \frac{1}{\gamma + (1-\gamma)/P + \gamma_{oh}}. \tag{17}$$

Note que si $\gamma \to 0$ y $\gamma_{oh} \to 0$,

$$S(P) \to P,$$

que es un *speedup* lineal.





Ej: Implemente versiones seriales y paralela de la suma de vectores. Estime el speedup haciendo los cálculos varias veces (calcular promedio y varianza).

Varianza de el producto de variables independientes X y Y:

$$\sigma_{XY}^2 = \sigma_X^2\sigma_Y^2 + \sigma_X^2\mu_Y^2 + \sigma_Y^2\mu_X^2 \tag{18}$$

Ej: Implemente versiones seriales y paralela del cálculo de la secuencia de fibonacci. Estime el speedup haciendo los cálculos varias veces (calcular promedio y varianza).