





Taller de Programación Paralela

Fernando R. Rannou
Departamento de Ingeniería Informática
Universidad de Santiago de Chile

April 15, 2015



Rendimiento

Rendimiento

Rendimiento computacional secuencial

Rendimiento

- *Rendimiento computacional secuencial* se refiere al costo computacional de un programa que se ejecuta en un procesador
- Los principales costos que se miden son
 1. **Tiempo**
 2. **Memoria**
- En la medida que el sistema tenga suficiente memoria para almacenar los datos del problema, el costo no tiene relevancia.
- Dicho de otra forma, si tengo memoria suficiente no gano nada quitándole memoria al sistema.
- Luego, con esta suposición, la única medida de interés en un sistema secuencial es el tiempo de ejecución.
- Soluciones secuenciales pueden ser comparadas usando análisis asintótico. Por ejemplo, un Quicksort $O(n^2)$ versus un Quicksort $O(n \log n)$.

Tiempo de ejecución

Rendimiento

- Sea τ el tiempo de ciclo del procesador; $f = \frac{1}{\tau}$ es la frecuencia del reloj (en Giga Hertz), o simplemente la velocidad del procesador.
- Sea I_c el número de instrucciones en un programa y
- CPI el número promedio de ciclos por instrucción. Luego

$$T = I_c \times CPI \times \tau$$

es el tiempo de ejecución

- Ejemplo: Suponga que un programa tiene 10^6 instrucciones, que se ejecutan en un procesador de 2.0 Ghz con un CPI de 14 ciclos por instrucción. Entonces, el tiempo de ejecución teórico es:

$$T = 10^6 \text{ Inst} \times 15 \text{ ciclos/Inst} \times \frac{1}{2 \times 10^9 \text{ ciclos/s}}$$

$$T = 7 \times 10^{-3} \text{ s}$$

- Note que I_c se refiere a instrucciones de máquina!

- **MIPS** (Millions of Instructions per Second): es una unidad de rendimiento que caracteriza la tasa de ejecución de un procesador.

$$\text{MIPS} = \frac{I_c}{T \times 10^6}$$

$$\text{MIPS} = \frac{f}{CPI \times 10^6}$$

- Note que MIPS es una medida que depende de factores como velocidad del procesador, conjunto de instrucciones, y tamaño del programa.
- Algunos vendedores reportam MIPS *Peak* de sus procesadores; el cual se obtiene eligiendo un programa con un CPI mínimo, sin importar que este programa no sirva para nada!
- MIPS (y MIPS peak) es considerada una medida inútil! No la use jamás para presentar el rendimiento de su máquina.

- **MFLOPS** (Millions of Floating-point Operations per Second)

$$MFLOPS = \frac{\text{Número de instrucciones de punto flotante en un programa}}{\text{Tiempo de ejecución}} \times 10^6$$

- Una operación punto flotante es una suma, resta, multiplicación o división entre números de precisión simple o doble (`float` y `double` en C)
- Se postula que MFLOPS es una medida más justa para medir el rendimiento de un procesador; *un mismo programa puede ejecutar un número distinto de instrucciones, pero siempre ejecutará el mismo número de operaciones de tipo flotante*
- Si embargo:
 - ◆ No todas las op. punto flotante demoran lo mismo
 - ◆ No siempre se genera el mismo número de op. en distintas arquitecturas
 - ◆ ¿Cómo medir operaciones como `sin()`, `log()`, etc?
 - ◆ Operaciones 64bit en procesadores 32bit
- MFLOPS es mejor que MIPS, pero aún muy imperfecta.



Benchmarks para reportar rendimiento de un computador

Rendimiento



- **Dhrystone:** Benchmark sintético de punto-fijo (procesador)
 - ◆ Mezcla de instrucciones de alto nivel sin operaciones flotantes
 - ◆ Varios tipos de instrucciones, tipos de datos y localidad de referencia
 - ◆ No hay llamados al sistema, no hay funciones de librería o llamados a subrutinas.
 - ◆ Dhrystone reporta MIPS
- **Whetstone:** Benchmark sintético de punto-flotante (procesador)
 - ◆ Benchmark en FORTRAN; incluye operaciones punto-fijo y flotante
 - ◆ Usa acceso a arreglos, llamados a subrutinas, funciones trigonométricas.
 - ◆ Reporta KWhetstones/s
- Ambos benchmarks no ayudan a predecir el rendimiento de programas usuarios
- Son sensibles al compilador



Otros benchmarks

Rendimiento



- **TPS:** (Transactions per second), usado para sistemas de transacciones
- **KLIPS:** (Kilo logic inferences per second), intenta medir el rendimiento de una máquina de inteligencia artificial
- **STREAM:** benchmark sintético para medir el ancho de banda de memoria (sostenido)

El Benchmark LINPACK

Rendimiento

- LINPACK (Linear Algebra Package) resuelve un sistema denso de ecuaciones lineales
 - ◆ Alto porcentaje de operaciones aritméticas de punto-flotante
 - ◆ Usa la librería BLAS (Basic Linear Algebra Subprograms)
 - ◆ Permite al usuario usar matrices de distintos tamaños.
- Para calcular la tasa de ejecución, usa $2n^3/3 + 2n^2$ operaciones donde n es el orden del sistema
- Actualmente usado para clasificar los computadores más veloces del mundo (www.top500.org)
- No mide el rendimiento completo de un sistema computacional; simplemente mide el rendimiento para resolver un problema particular (no sintético)

LINPACK (cont)

Rendimiento

- Reporta R_{\max} : GFLOPS sostenido, y R_{peak} : GFLOPS teórico.
- R_{peak} : cuenta el número de sumas y multiplicaciones (precisión completa) que pueden ser terminadas en un tiempo dado.
- Ejemplo: La Cray Y-MP/8 tiene un ciclo de reloj de 6 ns, durante el cual se puede completar una suma y una multiplicación. Luego

$$R_{\text{peak}} = \frac{2 \text{ op.}}{1 \text{ ciclo}} \times \frac{1 \text{ ciclo}}{6 \times 10^{-9} \text{ s}} = 333 \text{ MFLOPS}$$

- Si se usan 8 procesadores, $R_{\text{peak}} = 2.6 \text{ GFLOPS}$
- R_{\max} reporta el rendimiento para el problema más grande que se pueda correr en el computador.

Midiendo el tiempo de una aplicación

Rendimiento

Los siguientes son factores involucrados en la medición del tiempo:

- **Resolución:** Representa la precisión del sistema de medición. Por ejemplo, el *reloj de software* (clock) en un sistema operativo puede tener resolución de 0.001 segundos.
- **Exactitud:** Representa la cercanía del valor medido al verdadero. Se puede cuantificar en términos del error de la medición.
- **Granularidad:** Es la unidad de software que se quiere medir:
 1. granularidad gruesa: programa
 2. granularidad media: función
 3. granularidad fina: instrucción
- **Complejidad:** Nos dice cuán difícil es obtener la medición.

Ejemplo de métodos

Rendimiento

<i>Método</i>	<i>Resolución</i>	<i>Exactitud</i>	<i>Granularidad</i>	<i>Complejidad</i>
Reloj de pulsera	0.01 s	0.5 s	proceso	fácil
Comando date	0.02 s	0.2 s	proceso	fácil
Comando time	0.02 s	0.2 s	proceso	fácil
prof y gprof	10 ms	20 ms	funciones	moderada
clock()	10-30 ms	15-30 ms	instrucción	moderada
Contadores del procesador	0.4-4 μ s	1-8 μ s	instrucción	muy difícil

■ La elección de un método dependerá de factores tales como:

- ◆ Objetivo de la medición
- ◆ Herramientas disponibles
- ◆ Esfuerzo necesario
- ◆ etc.

Tiempo de reloj

Rendimiento

- El **tiempo de reloj** mide el tiempo total desde que se somete el proceso a ejecución hasta que termina.
- Se puede usar el reloj de pulsera o comandos como `time` y `date`

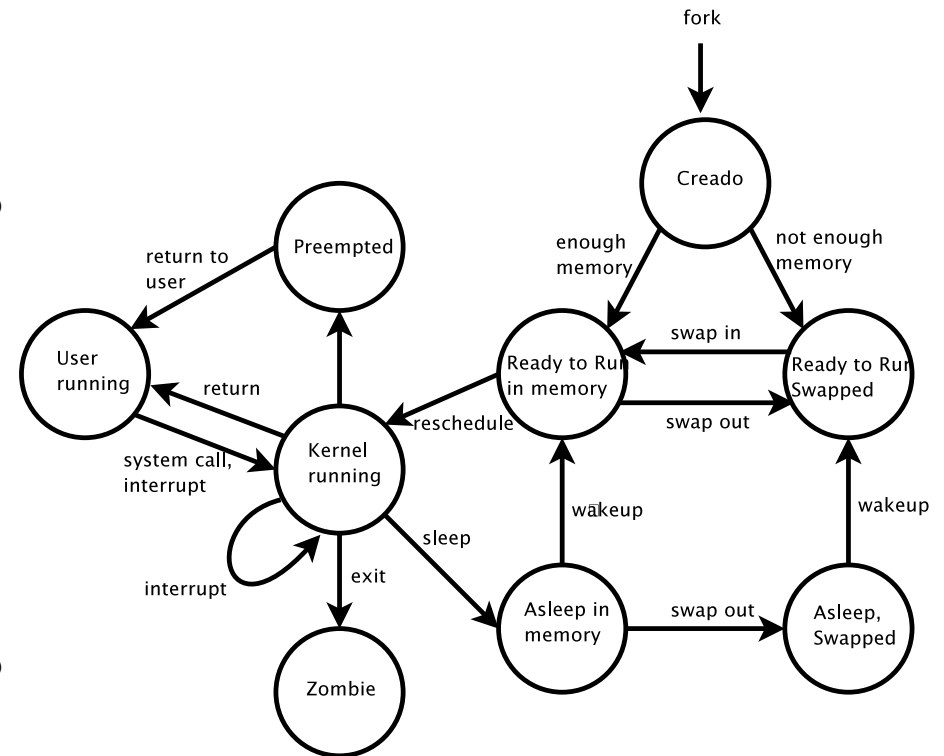
```
$ date >> logtime; ./zoom -i imagen1.raw -o ....; date >> logtime
$ cat logtime
Fri Apr 18 15:45:36 CLT 2008
Fri Apr 18 15:45:38 CLT 2008
$
$ time ./zoom -i imagen1.raw -o ...-z 64
real    0m4.098s
user    0m0.812s
sys     0m2.704s
```

- El reloj de pulsera y el comando `date` entregan el tiempo total considerando tiempos de procesador, tiempos de espera, tiempos de I/O, etc.

El comando time

Rendimiento

- time retorna el tiempo transcurrido (*elapsed time*) entre la invocación del proceso hasta el término.
- time retorna tres tiempos:
 1. **tiempo real** (real)
 2. **tiempo usuario** (usr)
 3. **tiempo sistema** (sys)
- El tiempo real es el tiempo transcurrido o total
- El tiempo usuario es el tiempo que el proceso se ejecuta en modo usuario
- El tiempo sistema es el tiempo que el proceso se ejecuta en modo kernel



```
$ time sleep 60
real    1m0.006s
user    0m0.000s
sys     0m0.000s
```

EL llamado al sistema times()

Rendimiento

```
#include <sys/times.h>
clock_t times(struct tms *buf);
```

- times() almacena los tiempos actuales de un proceso en la estructura struct tms apuntada por buf.

```
struct tms {
    clock_t tms_utime; /* user time */
    clock_t tms_stime; /* system time */
    clock_t tms_cutime; /* user time of children */
    clock_t tms_cstime; /* system time of children */
};
```

- El comando time retorna valores entregados por el llamado al sistema times()
 - ◆ el tiempo usuario es la suma de tms_utime y tms_cutime,
 - ◆ el tiempo del sistema la suma de tms_stime y tms_cstime.
- clock_t son *ticks* del system timer
- Para convertir a tiempo, necesitamos conocer el número de ticks por segundo



Ticks

Rendimiento



```
#include <unistd.h>
#include <stdio.h>
```

```
main() {
    printf("Ticks = %ld\n", sysconf(_SC_CLK_TCK));
}
```

```
$ gcc -o tikcs1 ticks1.c
$ ./ticks1
Ticks = 100
```


Ejemplo de times()

Rendimiento

```
#include <stdlib.h>
#include <sys/times.h>
#include <unistd.h>
#include <time.h>
#include <stdio.h>

struct tms mytime;

main() {
    ... // aqui va todo el codigo
    times(&mytime);

    printf("User = %f\n", (double) mytime.tms_utime/sysconf(_SC_CLK_TCK));
    printf("Sys  = %f\n", (double) mytime.tms_stime/sysconf(_SC_CLK_TCK));
}
```

```
$ time ./zoom -i ../../../../rostro183x150.raw -o junk.raw -f 183 -c 150 -z 64
User = 1.400000
Sys  = 2.640000
```

```
real    0m4.231s
user    0m1.400s
sys     0m2.724s
```

Otro ejemplo

Rendimiento

```
include <stdlib.h>
#include <sys/times.h>
#include <unistd.h>
#include <time.h>
#include <stdio.h>

struct tms mytime;
main() {
    sleep(300);
    times(&mytime);
    printf("User = %f\n", (double) mytime.tms_utime/sysconf(_SC_CLK_TCK));
    printf("Sys = %f\n", (double) mytime.tms_stime/sysconf(_SC_CLK_TCK));
}
```

```
$ ./times
User = 0.000000
Sys = 0.000000
```

El llamado al sistema `clock()`

Rendimiento

- `clock()` retorna el tiempo de procesador usado hasta el momento por el proceso

```
#include <time.h>
clock_t clock(void);
```

- El valor retornado es de tipo `clock_t`; para obtener el tiempo en segundos dividimos por `CLOCKS_PER_SEC`.
- Según POSIX, `CLOCKS_PER_SEC` debe ser 1000000, independientemente de la resolución real del reloj.
- Note que `clock()` no distingue entre tiempo usuario o del sistema.

Otros ticks???

Rendimiento

```
#include <unistd.h>
#include <time.h>
#include <stdio.h>

main() {
    printf("Ticks = %ld\n", CLOCKS_PER_SEC);
}

$ ./ticks2
Ticks = 1000000
```

Ejemplo de clock()

Rendimiento

```
#include <sys/times.h>
```

```
#include <time.h>
```

```
clock_t timestart, timeend;
```

```
main() {
```

```
    timestart = clock(); // registramos el tiempo hasta el momento
```

```
    ... // aqui va todo el codigo
```

```
    timeend = clock() // registramos el tiempo hasta el final
```

```
    printf("Total = %f\n", (double) (timeend-timestart)/((double)CLOCKS_PER_SEC);
```

```
}
```

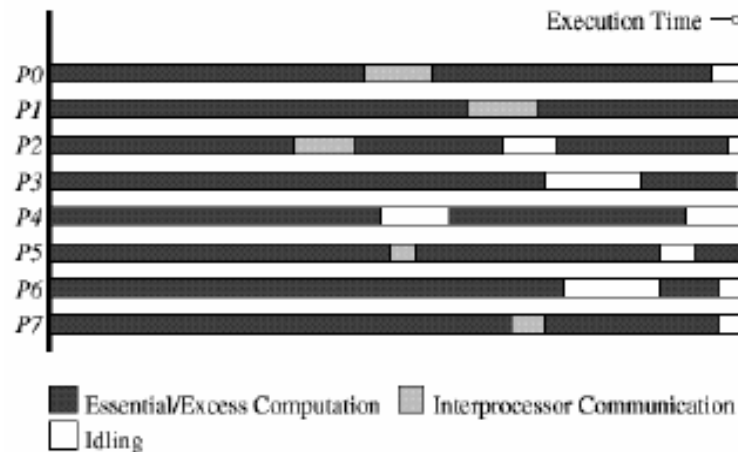
```
$ ./zoom -i ../../../../rostro183x150.raw -o junk.raw -f 183 -c 150 -z 64
```

```
Total = 4.040000
```

¿Y cómo medir el rendimiento paralelo?

Rendimiento

- El tiempo de ejecución de una aplicación paralela depende de muchos factores que no se encuentran en aplicaciones secuenciales
 - ◆ Latencia de comunicación entre procesadores
 - ◆ Balance de carga entre los nodos
 - ◆ Overhead de sincronización
 - ◆ Heterogeniedad de los procesadores
 - ◆ Adaptación del algoritmo a la topología de red



Rendimiento paralelo y Speedup

Rendimiento

- Cuando hablamos de análisis de rendimiento de una aplicación paralela, implicamos un análisis de un programa con respecto a una arquitectura paralela particular.
- Es la combinación de algoritmo y arquitectura la que define el *rendimiento del sistema*
- La medición del rendimiento de un sistema ha sido opacada por la necesidad de saber "cuánto más rápido corre nuestro programa en el computador paralelo"
- La forma más básica de medir dicho beneficio es el **speedup**:

$$\text{Speedup} = \frac{T_s}{T_p}$$

donde T_s es el tiempo de ejecución secuencial y T_p es el tiempo de ejecución paralelo.

- ¿Son T_s y T_p resultados del mismo algoritmo?
- ¿Son T_s y T_p resultados de la misma arquitectura?
- Existen al menos 5 definiciones de Speedup

Speedup asimptótico real

Rendimiento

$$S_{\text{AsimptoticoReal}}(n) = \frac{t_{\text{serial}}^{\text{best}}(n)}{t_{\text{parallel}}^Q(n)}$$

donde $t_{\text{serial}}^{\text{best}}(n)$ es la complejidad asimptótica del mejor algoritmo secuencial conocido para un problema de tamaño n , y $t_{\text{parallel}}^Q(n)$ es la complejidad del algoritmo paralelo Q asumiendo que dispone de un número infinito de procesadores.

- También conocido como **Speedup analítico**
- Si se usan tiempos reales de ejecución se llama **Speedup medido**
- Independiente del número de procesadores
- Cambia con el tiempo, a medida que se encuentran mejores algoritmos secuenciales

Ejemplo: Existe un algoritmo paralelo de multiplicación de matrices que demora $O(\log n)$ en un computador hipercubo de $n^3 / \log n$ procesadores. Se ha demostrado que este tiempo es el mejor, aún cuando se usen más procesadores. ¿Pero cuál es el mejor algoritmo secuencial para multiplicar matrices? Asumiendo $O(n^\alpha)$, obtenemos

$$S_{\text{AsimptoticoReal}}(n) = \frac{n^\alpha}{\log n}$$

Speedup absoluto

Rendimiento

$$S_{\text{Absoluto}} = \frac{\text{tiempo para resolver } I \text{ usando el mejor algoritmo secuencial y el procesador más veloz}}{\text{tiempo para resolver } I \text{ usando programa } Q \text{ y } P \text{ procesadores}}$$

- En primera instancia, esta medida pareciera muy importante, pues mide cuánto más rápido mi solución paralela es respecto de la mejor solución serial (algoritmo + procesador)
- Es posible disponer del mejor algoritmo (hasta ahora conocido) para un problema, pero ¿disponemos del procesador más veloz del mundo?
- Los computadores en la lista de los 100 más veloces del mundo usan procesadores “lentos”; por lo tanto es posible medir speedup absoluto menor que 1.
- Ejemplo: se encuentra en la literatura¹ que un nCube1 de 64 procesadores se demora seis veces más que un Cray2 monoprocesador en realizar una operación de *template matching* sobre una imagen de 256×256 .

¹S. Ranka and S. Sahni, Image template matching on MIMD hypercube multicomputers, *J. Parallel and Distributed Computing*, 10, 1990, pp. 79-84

Speedup real

Rendimiento

$$S_{\text{Real}} = \frac{\text{tiempo para resolver } I \text{ usando el mejor algoritmo secuencial y 1 procesador}}{\text{tiempo para resolver } I \text{ usando programa } Q \text{ y } P \text{ procesadores}}$$

- Note que:
 - ◆ Speedup asintótico real usa infinitos procesadores
 - ◆ Speedup real usa P procesadores
- Es posible calcular el speedup real usando complejidades.
Ejemplo: El algoritmo secuencial más veloz para sumar dos matrices de $n \times n$ es $O(n^2)$; si el programa paralelo toma $O(n^2/P)$ usando $P \leq n^2$ procesadores con memoria compartida, el **Speedup real analítico** es $O(P)$.
- **Speedup real medido** se obtiene midiendo los tiempos de ejecución del mejor algoritmo secuencial y del programa paralelo.
- Muchas veces es imposible medir el speedup real en un sistema de memoria distribuida, pues la memoria disponible en un nodo no es suficiente.

Speedup real (cont)

Rendimiento

- También es posible calcularlo usando carga de trabajo (incluye comunicación e I/O)
En el ejemplo anterior, leer dos matrices y escribir el resultado demora $3n^2$ operaciones.
Si la velocidad del procesador es V , entonces la suma secuencial es n^2/V . El computador paralelo toma $2n^2/(VP)$ en sumar, comunicar y overhead. Luego el speedup real es:

$$\frac{3n^2 + n^2/V}{3n^2 + n^2/(VP)} = \frac{3 + 1/V}{3 + 1/(VP)}$$

- A medida que se usa más procesadores, es posible que la solución paralela use más comunicación (y overhead) y el speedup real comenzaría a decaer.
- Además, note que en el caso anterior, si $P > 2$ el speedup aumenta si usamos procesadores más lentos.
- Sin embargo, el tiempo de ejecución paralelo disminuye.

Mejorar el speedup real debiera ser un objetivo secundario a reducir el tiempo de ejecución paralelo

Speedup relativo

Rendimiento

$$S_{\text{Relativo}}(I, P) = \frac{\text{tiempo para resolver } I \text{ usando programa } Q \text{ y } 1 \text{ procesador}}{\text{tiempo para resolver } I \text{ usando programa } Q \text{ y } P \text{ procesadores}}$$

- No siempre es posible estar seguro que uno tiene el mejor algoritmo secuencial; speedup relativo usa el mismo programa paralelo Q y un procesador para reportar el tiempo secuencial.
- Sobreestima el speedup cuando el programa secuencial es ineficiente cuando corre en un procesador
- Por limitaciones de memoria, a veces es imposible correr programas en un sólo procesador
- Favorece programas paralelos que son ineficientes cuando se ejecutan en 1 procesador

Fallas en Speedup relativo

Rendimiento

- Considere un programa para multiplicar dos matrices de $n \times n$ en un computador paralelo de P procesadores conectados como periféricos a un procesador maestro (sin I/O). Se sabe que el tiempo para transferir las matrices a los procesadores y recuperar el resultado toma $3n^2$.

El tiempo para multiplicar las matrices es $n^3/(VP)$, donde V es la velocidad de un procesador (en términos de operaciones por segundo) y $P \leq n^2$. Luego el speedup relativo es

$$S_{\text{relativo}} = \frac{n^3/V}{3n^2 + n^3/(VP)} = \frac{n}{3V + n/P}$$

- Plop! Speedup mejora si usamos procesadores más lentos...
- Suponga que la multiplicación es ineficiente, es decir hace más trabajo de lo necesario.

$$S_{\text{relativo}} = \frac{kn^3/V}{3n^2 + kn^3/(VP)} = \frac{n}{3V/k + n/P}$$

la cual es una función creciente en k