Benchmarks Sintéticos

Fernando Emmanuel Frati

III LIDI
Facultad de Informática

Curso Programación en Cluster, 2010

Outline

Outline

Definition

Es una técnica utilizada para medir el rendimiento de un sistema o componente del mismo. Es el resultado de la ejecución de un programa informático o un conjunto de programas en una máquina, con el objetivo de estimar el rendimiento de un elemento concreto o la totalidad de la misma, y poder comparar los resultados con máquinas similares.

- Sintéticos (dhrystone o whetstone)
- Aplicaciones (SPEC)

Outline

El tiempo es la unidad de medida preferida cuando se comparan varios procesadores. El tiempo de ejecución de un programa se puede dividir en:

- **Tiempo de respuesta** Es el tiempo necesario para completar una tarea, incluyendo los accesos al disco, a la memoria, las actividades de E/S y los gastos del S.O. Es el tiempo que percibe el usuario.
  - **Tiempo de CPU** Es el tiempo que tarda en ejecutarse un programa, sin tener en cuenta el tiempo de espera debido a la E/S o el tiempo utilizado para ejecutar otros programas. Se divide en:
    - **Tiempo de CPU utilizado por el usuario**. Es el tiempo que la CPU utiliza para ejecutar el programa del usuario.
    - **Tiempo de CPU utilizado por el S.O.** Es el tiempo que el S.O. emplea para realizar su gestión interna.

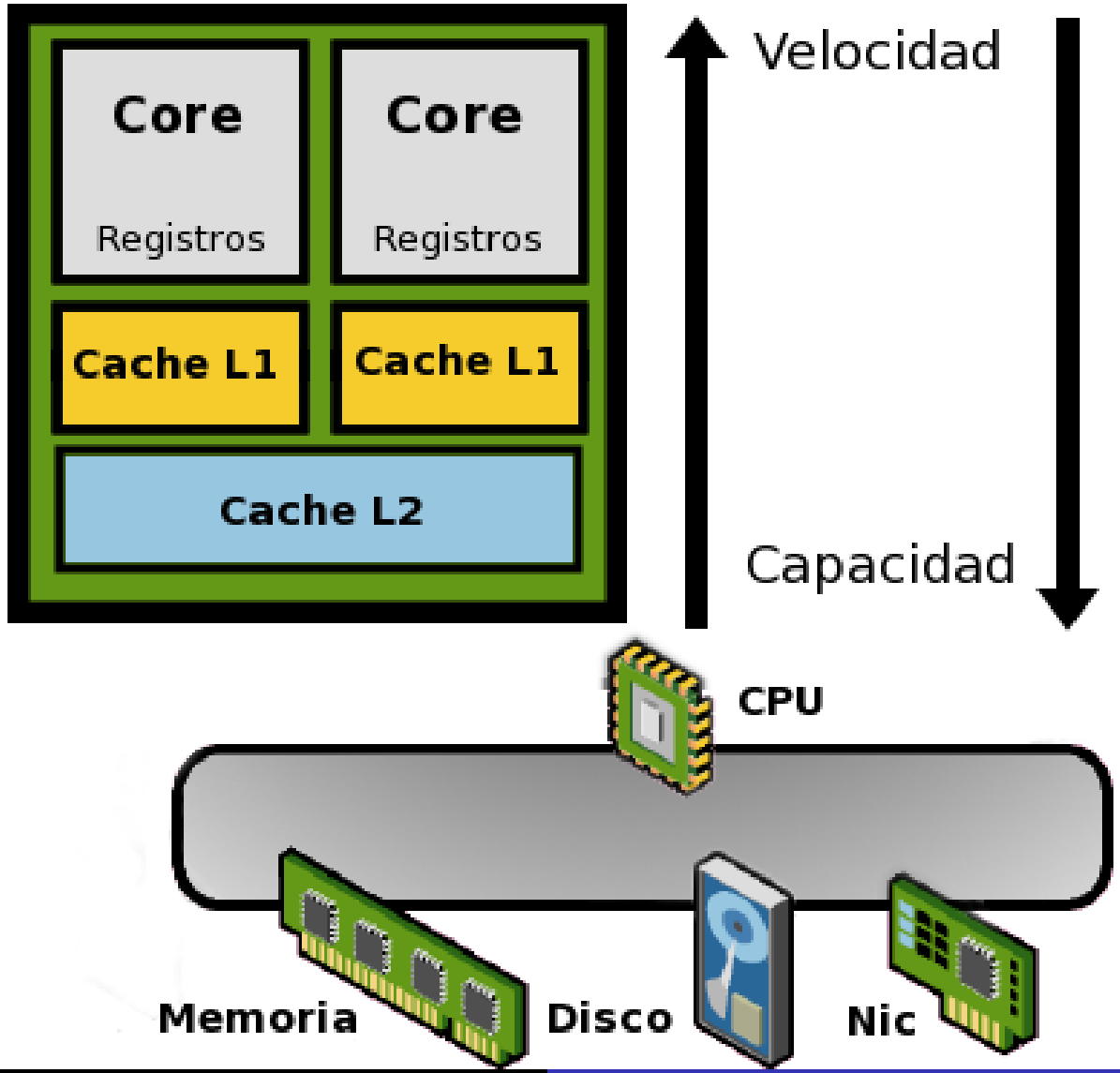
La salida se formatea utilizando una cadena de formato que se puede especificar utilizando la opción -f o la variable de entorno TIME.
Cuando ejecutamos "time" en la bash, estamos ejecutando la librería time de bash, es decir, la primera que viene en el manual del "man time". La segunda parte que dice todas las opciones útiles para poder realizar scripts que se llama GNU VERSION se encuentra en el siguiente path:
/usr/bin/time
Para sacar sólo el tiempo real en segundos de un comando usaremos:
/usr/bin/time -f%e comando

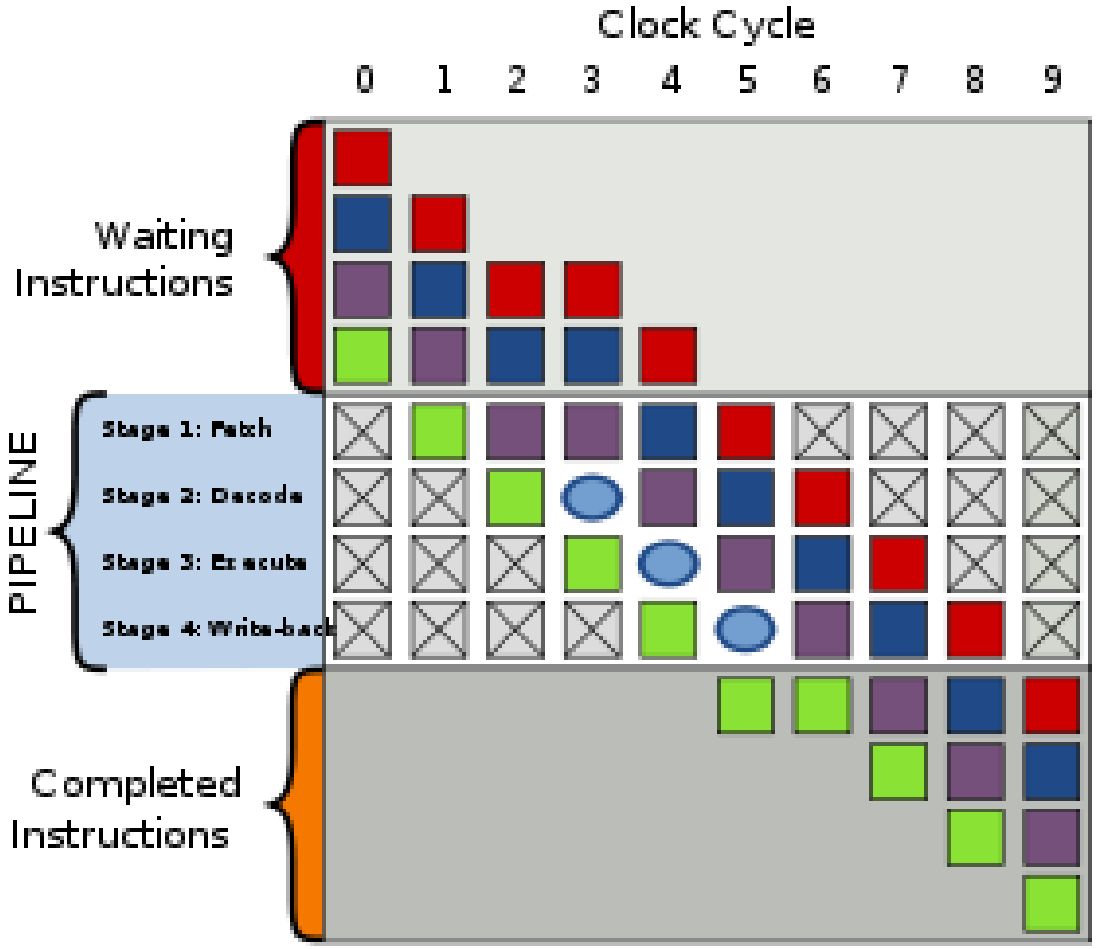
El formato se interpreta en el modo usual de printf. Los caracteres habituales se copian directamente, tabulación, nueva línea y barra invertida se escapan usando \t, \n y \\.
A continuación se presentan las conversiones:

- %E Tiempo real transcurrido (en [horas:]minutos:segundos).
- %e Tiempo real transcurrido (en segundos).
- %S Número total de segundos de CPU que el proceso consumió en modo de núcleo.
- %U Número total de segundos de CPU que el proceso consumió en modo de usuario.
- %P Porcentaje de CPU que recibió este trabajo, calculado como (%U + %S) / %E.

Ejemplo: se puede establecer la variable de entorno TIME para que muestre el tiempo de acuerdo a nuestras necesidades:
export TIME="Tiempo Total: % e\nTiempo de uso de CPU: % U + % S (Usuario + Sistema)\n Porcentaje de uso de CPU: % P"

Outline


Core
Registros
Core
Registros
Cache L1
Cache L1
Cache L2
Velocidad
Capacidad
CPU
Memoria
Disco
Nic


Clock Cycle
0 1 2 3 4 5 6 7 8 9
Waiting Instructions
PIPELINE
Stage 1: Fetch
Stage 2: Decode
Stage 3: Execute
Stage 4: Write-back
Completed Instructions

Conjunto	Descripción
x86	Conjunto de instrucciones de propósito general y de operaciones enteras.
x87	Extiene el soporte para instrucciones en coma flotante.
MMX	Primer conjunto de instrucciones SIMD enteras pensadas para rendimiento multimedia.
SSE	Extiende MMX para soportar números en coma flotante de simple precisión.
SSE2	Soporta flotantes de doble precisión y enteros de 128 bits.
SSE3	Extensión para soportar procesos (threads).
SSE4	Extensión a instrucciones que no son específicamente para multimedia.

pipeline + conjuntos de instrucciones = microarquitectura

Procesador objeto de estudio:

- Intel Core Duo T3200
- Frecuencia 1660 MHz

Instrucciones a utilizar:

Instrucción	Latencia	Throughput	Conjunto
addl	1	0,5	x86
cmpl	1	0,5	x86
movl*	1	0,5	x86
jle**	1	0,5	x86
fadd	3	1	x87

Outline

Algoritmo muy simple. Cantidad de iteraciones arbitrarias:

```
int main(int argc, char *argv){
    int c, d;
    d=0;
    do{
        c=0;
        do
            c=c+1;
        while(c < 10000000);
        d++;
    } while(d < 1000);
    return 0;
}
```

Tiempo de ejecución: 49,6 segundos.

Para averiguar el número de instrucciones:
[usuario@host ~]$gcc mips.c -S -fverbose-asm

- **-S**: crear un archivo .s, que es el codigo en asembler equivalente al algoritmo en C
- **-fverbose-asm**: insertar comentarios (como los nombres de las variables) en el codigo asembler generado.

$$MIPS = \frac{NI}{Tiempo \times 10^6} = \frac{30000004000}{49{,}6 \times 10^6} = 604{,}83$$

$$NI = (3 \times 10000000 + 4) \times 1000 = 30000004000$$

$$Ciclos = (3 \times (0{,}5\ ciclos) \times 10000000 + 4 \times (0{,}5\ ciclos)) \times 1000 = 15000002000$$

Tiempo que debería tardar:

$$Segundos = \frac{MCiclos}{MHz}$$

$$MCiclos = \frac{15000002000}{1000000} = 15000{,}002$$

$$Segundos = \frac{15000{,}002}{1660} = 9{,}036145783$$

MIPS Teórico:

$$MIPS' = \frac{NI}{Tiempo \times 10^6} = \frac{30000004000}{9{,}036145783 \times 10^6} = 3320{,}000000049$$

$$\%Rendimiento = \frac{MIPS}{MIPS'} \times 100 = \frac{604{,}83}{3320{,}000000049} \times 100 = 18{,}21\,\%$$

```
int main(int argc, char *argv){
    asm(
        "movl $0,%eax\n"
    ".DO1:\n\t"
        "movl $0,%ebx\n"
    ".DO2:\n\t"
        "addl $1,%ebx\n\t"
        "cmpl $9999999,%ebx\n\t"
        "jle .DO2\n\t"
        "addl $1,%eax\n\t"
        "cmpl $999,%eax\n\t"
        "jle .DO1\n\t"
    );
    return 0;
}
```

Tiempo de ejecución: 9,15 segundos!!!

$$MIPS = \frac{NI}{Tiempo \times 10^6} = \frac{30000004000}{9{,}15 \times 10^6} = 3278{,}68$$

$$\%Rendimiento = \frac{MIPS}{MIPS'} \times 100 = \frac{3278{,}68}{3320{,}000000049} \times 100 = 98{,}75\,\%$$

Outline

```
int main(int argc, char *argv){
    asm(
        "fld1\n\t"
        "fld1\n\t"
        "movl $0,%eax\n"
    ".DO1:\n\t"
        "movl $0,%ebx\n"
    ".DO2:\n\t"
        "fadd %st(1),%st(0)\n\t"
        "addl $1,%ebx\n\t"
        "cmpl $9999999,%ebx\n\t"
        "jle .DO2\n\t"
        "addl $1,%eax\n\t"
        "cmpl $999,%eax\n\t"
        "jle .DO1\n\t"
    );
    return 0;
}
```

- fld1: carga el valor 1 en la la pila
- fadd: realiza la suma st(1) + st(0)

Tiempo de ejecución: 20,24 segundos

$$NI = (4 \times 10000000 + 4) \times 1000 = 40000004000$$

$$MFLOPS = \frac{NI}{Tiempo \times 10^6} = \frac{40000004000}{20{,}24 \times 10^6} = 1976{,}28$$

$$NI = ((1x87 + 3x86) \times 10000000 + 4x86) \times 1000$$

- Menos del 25 % de las instrucciones son x87.
- Nuevo algoritmo, con más cantidad de instrucciones x87 y menos iteraciones en bucle externo
- Nuevo tiempo de ejecución: 18,22 segundos

$$NI = (103 \times 10000000 + 4) \times 10 = 10300000040$$

$$MFLOPS = \frac{NI}{Tiempo \times 10^6} = \frac{10300000040}{18{,}22 \times 10^6} = 565{,}31$$

$$Ciclos = ((99 \times 3\ ciclos + 1 \times 2\ ciclos + 3 \times 0{,}5\ ciclos) \times 10^7 + 4 \times 0{,}5\ ciclos) \times 10$$

$$Ciclos = 30050000020$$

Tiempo que debería tardar:

$$Segundos = \frac{MCiclos}{MHz}$$

$$MCiclos = \frac{30050000020}{1000000} = 30050,00002$$

$$Segundos = \frac{30050,00002}{1660} = 18,102409651$$

MFLOPS Teórico:

$$MFLOPS' = \frac{NI}{Tiempo \times 10^6} = \frac{10300000040}{18,102409651 \times 10^6} = 568,985026777$$

$$\%Rendimiento = \frac{MFLOPS}{MFLOPS'} \times 100 = \frac{565,31}{568,985026777} \times 100 = 99,35\,\%$$

Conclusiones

- Los benchmarks sintéticos pueden ayudarnos a estimar el rendimiento de los componentes de nuestro sistema.
- La optimización de código para explotar el máximo rendimiento requiere mucho trabajo y está muy comprometida con la arquitectura.