



Medición del Rendimiento de Computadores

Motivación:

- ✓ ¿Qué medidas de tiempo se utilizan para evaluar el rendimiento?
- ✓ ¿Cómo definimos el tiempo de cpu y el rendimiento del procesador?
- ✓ ¿Qué parámetros condicionan el tiempo de cpu?
- ✓ ¿Qué medidas de productividad se utilizan para evaluar el rendimiento?
- ✓ ¿Qué ventajas e inconvenientes presentan dichas medidas?
- ✓ ¿Qué tipos de programas se utilizan para evaluar el rendimiento?



Medición del Rendimiento de Computadores

- Introducción
- Tiempo de ejecución. Rendimiento del Procesador
- Medidas de Productividad
 - ✓ MIPS
 - ✓ MFLOPS
- Programas de evaluación (*Benchmarks*)



Medición del Rendimiento

- ¿Por qué se necesita medir el rendimiento?
 - ✓ comparación del hardware de las máquinas
 - ✓ comparación del software de las máquinas (compiladores)
- Propósito final:
 - ✓ tomar decisiones de compra
 - ✓ desarrollar nuevas arquitecturas
- ¿Basta con una sola medida?
 - ✓ ¿es más rápida una máquina de 2 GHz que otra de 1,5 GHz?
- Medidas utilizadas:
 - ✓
 - ✓



Tiempo de Ejecución

- Tiempo de Respuesta (transcurrido, de reloj, *elapsed time*)
 - ✓ incluye todo (procesador, accesos a memoria, E/S, S.O., multiprogramación)
 - ✓ en un sistema descargado da idea del rendimiento del computador o de sistema
 - ✓ útil, pero no muy adecuado para efectuar comparaciones
- Tiempo de CPU
 - ✓ no incluye E/S ni tiempo ejecutando otros programas (multiprogramación)
 - ✓ $T_{cpu} = T_{cpu}(\text{usuario}) + T_{cpu}(\text{sistema})$
 - ✓ $T_{cpu}(\text{sistema}) = \text{tiempo ejecutando código del S.O.}$
- $T_{cpu}(\text{usuario})$
 - ✓ tiempo dedicado a la ejecución de código de “nuestro” programa
 - ✓ da idea del Rendimiento del Procesador o Rendimiento de CPU
 - ✓ esta influido por



Comando “time” de Unix

- Resume los tiempos consumidos en la ejecución de un programa

- Formato:

\$time programa

- Ejemplo de salida:

```
90.7u 12.9s 2:39 65% (shell csh)
real 2m 39s          (shell bash)
user 1m 30.7s
sys 0m 12.9s
```

$T_{cpu}(usuario) = 90,7 \text{ s}$. $T_{cpu}(sistema) = 12,9 \text{ s}$. $T_{transcurrido} = 159 \text{ s}$.

$65\% = \% \text{ del } T_{cpu} \text{ sobre el } T_{transcurrido} =$

$100\% - 65\% = 35\% = \% \text{ de tiempo esperando por E/S o ejecutando otros programas}$



Ciclo de Reloj

- El rendimiento se calcula en función del ciclo de reloj

$$\frac{\text{segundos}}{\text{programa}} = \frac{\text{ciclos}}{\text{programa}} \times \frac{\text{segundos}}{\text{ciclo}}$$

- El tiempo se puede medir en ciclos de reloj



- ✓ Tiempo de ciclo (**T**) = tiempo entre dos eventos de reloj (período de reloj)
- ✓ Frecuencia de reloj (**f**) = ciclos / segundo
- ✓
- ✓



Ciclos por Instrucción

- ¿Son iguales el número de instrucciones y el de ciclos?, **NO**
 - ✓ el número de ciclos depende de la organización hardware
 - ✓ el número cambia para cada procesador
- ¿Consumen todas las instrucciones igual n° de ciclos?, **NO**
 - ✓ las multiplicaciones/divisiones consumen mas que las sumas/restas
 - ✓ las instrucciones flotantes consumen mas que las enteras
 - ✓ las que acceden a memoria consumen mas que las que acceden a registros
- Ciclos por Instrucción (**CPI**)
 - ✓ número de ciclos promedio consumido por las instrucciones de un programa
 - ✓ permite comparar organizaciones alternativas que soportan una misma ISA
 - ✓ Si $NI = n^{\circ}$ de instrucciones del programa \Rightarrow



Rendimiento del Procesador

NI = nº de instrucciones del programa

CPI = ciclos por Instrucción

T = tiempo de ciclo

nº total de ciclos = NI x CPI

T_{cpu} (usuario) = nº total de ciclos x tiempo de ciclo = (NI x CPI) x T

T_{cpu} = T x ∑ NI_i x CPI_i (i = tipo de instrucción)

CPI = ∑ CPI_i x Fi (Fi = frecuencia de aparición de instrucciones i)

NI = función(Compilador, Lenguaje máquina)

CPI = función(Lenguaje máquina, Organización)

T = función(Organización, Tecnología)



Rendimiento del Procesador

$$T_{CPU} = NI \times CPI \times T$$

- La mejora de uno de los parámetros puede empeorar los otros, de forma que no disminuya el tiempo de ejecución
- Ejemplo: Mejora en el juego de instrucciones $\Rightarrow NI \downarrow$

Si $NI \downarrow$ y $T = cte.$ \Rightarrow puede ocurrir que $CPI \uparrow$ (inst. + complejas)

$T_{cpu} \sim NI \downarrow \times CPI \uparrow \Rightarrow$

Si $NI \downarrow$ y $CPI = cte.$ \Rightarrow puede ocurrir que $T \uparrow$ (inst. + complejas)

$T_{cpu} \sim NI \downarrow \times T \uparrow \Rightarrow$



Problema 2.1

1) NI de transferencia afectadas por cada 100 de programa:

NI aritméticas que desaparecen:

nuevo NI por cada 100 del antiguo:

nuevos NI por cada 93 de programa:

Control:

Aritméticas:

Transferencia:

nuevos % de utilización:

Control:

Aritméticas:

Transferencia:

nuevos CPI = antiguos CPI

$CPI_{nuevo} =$

2) $CPI_{antiguo} =$

$$T_{CPU\ antiguo} = NI \times CPI_{antiguo} \times T$$

$$T_{CPU\ nuevo} = 0,93NI \times CPI_{nuevo} \times 1,1T$$

$$G = T_{CPU\ antiguo} / T_{CPU\ nuevo} =$$



Medidas de Productividad



La única medida fiable es el tiempo de ejecución programas reales
Dos aspectos: Rendimiento del computador, *Rendimiento del procesador*



MIPS (nativos)

- **Millones de Instrucciones Por Segundo**

$$\text{MIPS} = \frac{\text{NI}}{\text{Tejecucion} \times 10^6} = \frac{\text{NI}}{\text{TCPU} \times 10^6} = \frac{1}{\text{CPI} \times T \times 10^6} = \frac{f}{\text{CPI} \times 10^6}$$

- Mayor velocidad implica mayor valor de MIPS (*a priori*)

$$\text{Tejecucion} = \frac{\text{NI}}{\text{MIPS} \times 10^6}$$



MIPS (nativos)

- **Inconvenientes:**

- ✓ MIPS depende del juego de instrucciones
 - ⇒ difícil comparar máquinas con JI diferentes
- ✓ MIPS depende del programa (de su mezcla de instrucciones)
- ✓ MIPS puede llegar a variar inversamente al rendimiento
 - ⇒ medida no consistente (Ejemplo: incorporación de FPU → $NI \neq cte$)
 - solución:



MIPS relativos

- Se elige una máquina de referencia a la que se le da el valor de referencia MIPS=1

$$\frac{\text{Referencia}}{\text{Tejecucion}} = \frac{\text{MIPS relativos}}{\text{MIPS referencia}}$$

- Modelo VAX 11/780 (1978) \Rightarrow MIPS referencia = 1 (~ 10E6 Inst./seg.)
- Referencia (VAX 11/780) / Tejecucion = G = MIPS relativos

Ventajas:

- ✓ resuelve el problema de la inconsistencia de los MIPS nativos

Inconveniente:





MFLOPS (nativos)

- *Millions of FLoating Point Operations Per Second*

$$\text{MFLOPS} = \frac{\text{Nº Operaciones Flotantes}}{\text{Tejecucion} \times 10^6}$$

- Aplicable a programas con Operaciones Flotantes:
 - ✓ sumas, restas, multiplicaciones, divisiones
- Simple precisión o Doble precisión



MFLOPS (nativos)

- **Inconvenientes:**

- ✓ MFLOPS dependen de si se dispone o no de FPU (*Floating Point Unit*)
- ✓ MFLOPS dependen de si están o no soportadas funciones complejas (multiplicación, división, raíz cuadrada, seno, coseno, etc.)
- ✓ MFLOPS dependen de la mezcla de I. enteras y flotantes del programa
- ✓ MFLOPS dependen de la mezcla de operaciones flotantes (operaciones mas complejas tienen un mayor tiempo de ejecución)
solución:



MFLOPS normalizados

- Se da un peso a cada tipo de operación flotante proporcional a su complejidad
- Se utilizan tablas de conversión de MFLOPS nativos a normalizados

Ejemplo:

Tipo de operación (i)	Nº Op.norm.
+, -, *, COMP	1
/, SQRT	4
EXP, SIN, COS	8

$$N^{\circ} \text{ total de Op. normalizadas} = \sum_i (N^{\circ} \text{Op.})_i \times (N^{\circ} \text{Op.norm.})_i$$

MFLOPS normalizados =



Medición del Rendimiento

- **La única medida objetiva es el Tiempo de Ejecución**
- **Cualquier otra medida ofrece una visión parcial**
 - ✓ n^o de instrucciones por programa
 - ✓ n^o de ciclos por programa
 - ✓ n^o de ciclos por instrucción
 - ✓ n^o de segundos por ciclo
 - ✓ n^o de ciclos por segundo
 - ✓ n^o de instrucciones por segundo



Resumen de Vocabulario

NI	número de instrucciones (<i>máquina</i>)
CPI	(<i>ciclos por instrucción</i>)
Tejecucion	tiempo de ejecución (<i>segundos por programa</i>)
T	tiempo de ciclo (<i>segundos por ciclo</i>)
f	frecuencia de reloj (<i>ciclos por segundo</i>)
MIPS	(<i>millones de instrucciones por segundo</i>)
MFLOPS	(<i>millones de operaciones flotantes por segundo</i>)



Programas de Evaluación (*Benchmarks*)

- Carga ideal para la evaluación:
 - ✓ Programas Usuario + Comandos S.O. (Carga Trabajo, *Workload*)
- Problema:
 - ✓ No disponibilidad de los computadores para correr la Carga de Trabajo
- Alternativa:
 - ✓ Cargas neutrales ejecutadas por los fabricantes u organismos evaluadores
- Tipos de programas de evaluación (*benchmarks*):
 - ✓
 - ✓
 - ✓



Benchmarks sintéticos

- Simulan la frecuencia de instrucciones y operandos de un abanico de programas reales (código artificial)
- Ejemplos:
 - ✓ Dhrystone (código entero)
 - ✓ Whetstone (código flotante)
- Inconvenientes:
 - ✓ optimizaciones del fabricante solo para la mejora de ese tipo de programas
 - ✓ compiladores detectores de benchmarks
 - ⇒
 - ✓ su pequeño tamaño da lugar a un rendimiento óptimo del sistema de memoria
 - ⇒

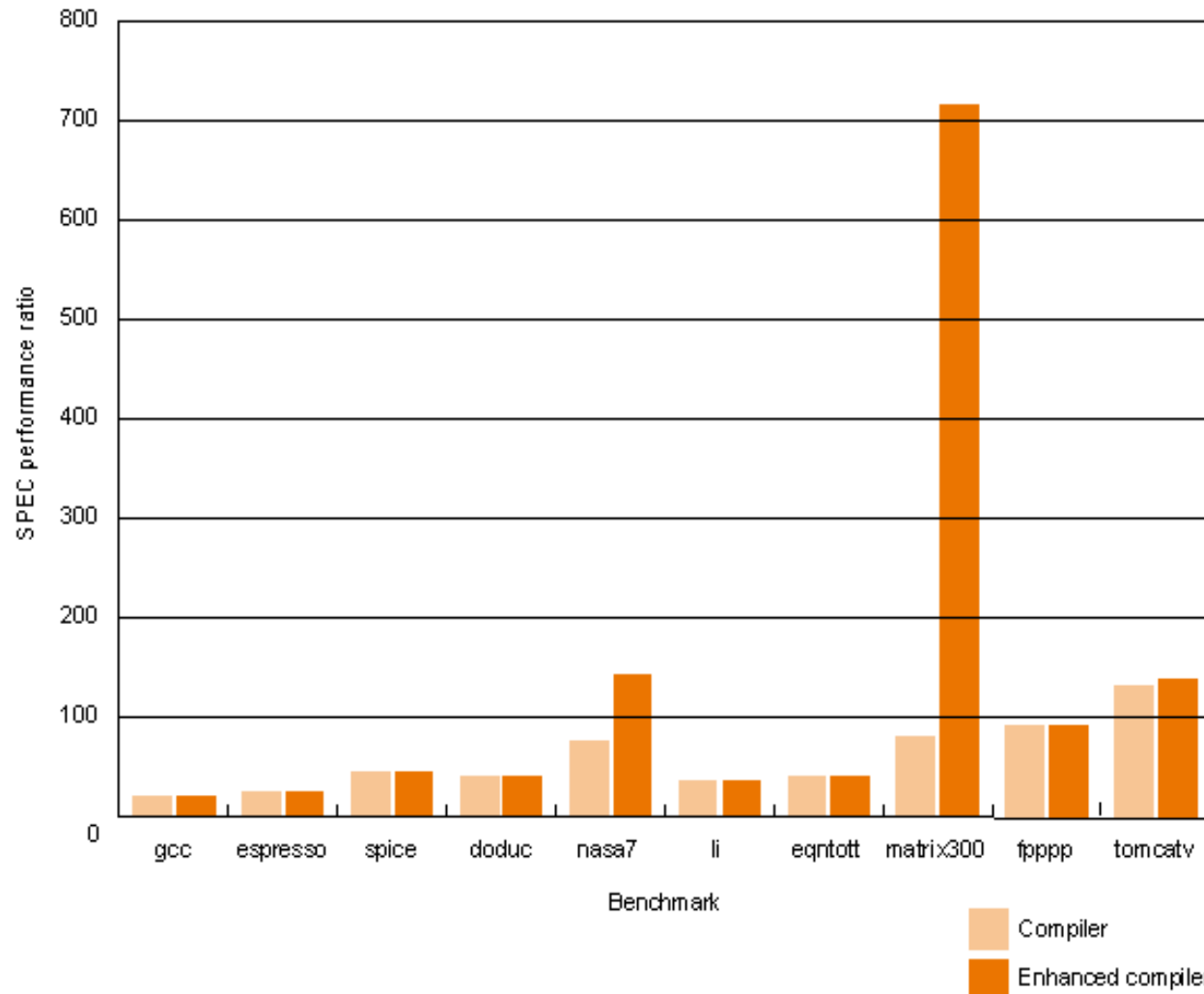


Programas de tamaño reducido

- Pequeños programas de entre 10 y 100 líneas con resultado conocido
- Incluyen normalmente bucles, repitiéndose mucho algunas instrucciones
- Fáciles de simular durante el diseño de la máquina (compilador no disponible)
- Fáciles de estandarizar
- Ejemplos:
 - ✓ programas de ordenación
 - ✓ multiplicación de matrices, etc.
 - ✓ **Flops**: resolución de integrales
 - ✓ **Linpack**: resolución de sistemas de ecuaciones lineales
- Inconvenientes:
 - ✓ optimizaciones del fabricante solo para la mejora de ese tipo de programas
 - ✓ rendimiento óptimo del sistema de memoria \Rightarrow



Optimizaciones específicas para *Benchmarks*





Aplicaciones reales

- Programas de uso común que forman parte de las cargas de trabajo
 - ✓ Compiladores y Editores de texto
 - ✓ Bases de datos y Hojas de cálculo
 - ✓ Programas científicos
- Ejemplos: Conjunto de programas **SPEC** y *benchmark* **SYSmark** (PCs)
 - ✓ SPEC =
 - ✓ Programas consensuados por el conjunto de fabricantes
 - ✓ Los mas utilizados para medir rendimiento y eficiencia del compilador
 - ✓ Actualizaciones continuas: SPEC 92, SPEC 95, SPEC 2000, SPEC 2006
 - ✓ SYSmark:
 - ✓



Problema 2.2

- 1) MIPS con copro = $f_{\text{CPU}} / (\text{CPI con copro} \times 10\text{E}6) =$
MIPS sin copro = $f_{\text{CPU}} / (\text{CPI sin copro} \times 10\text{E}6) =$

- 2) NI con copro = MIPS con copro $\times 10\text{E}6 \times \text{Tej con copro} =$
NI sin copro = MIPS sin copro $\times 10\text{E}6 \times \text{Tej sin copro} =$

- 3) NI con copro = $\quad \quad \quad = N \text{ Iter} \times (\text{NI enteras básicas} + \text{NI flotantes})$
 \Rightarrow NI enteras básicas =
NI sin copro = $\quad \quad \quad = N \text{ Iter} \times (\text{NI enteras básicas} + \text{NI enteras emulación})$
 \Rightarrow NI enteras emulación =
NI medio enteras / flotante = NI enteras emulación / NI flotantes =

- 4) MFLOPS nativos = $N^{\circ} \text{ Operaciones flotantes} \times N \text{ Iter} / (\text{Tej con copro} \times 10\text{E}6) =$
 $N^{\circ} \text{ Op. Normalizadas} = (n^{\circ} \text{ sum} + n^{\circ} \text{ res} + n^{\circ} \text{ mul} + n^{\circ} \text{ conv} + n^{\circ} \text{ comp}) \times 1 + n^{\circ} \text{ div} \times 4 =$
MFLOPS normalizados = $N^{\circ} \text{ Op. Normalizadas} \times N \text{ Iter} / (\text{Tej con copro} \times 10\text{E}6) =$

- 5) $N^{\circ} \text{ Op. en funciones} = N^{\circ} \text{ total de Op. flotantes} - N^{\circ} \text{ basico de Op. flotantes} =$
 $N^{\circ} \text{ invocaciones función} = n^{\circ} \text{ atan} + n^{\circ} \text{ sin} + n^{\circ} \text{ cos} + n^{\circ} \text{ sqrt} + n^{\circ} \text{ exp} + n^{\circ} \text{ log} =$
 $N^{\circ} \text{ medio Op. / función} = N^{\circ} \text{ Op. en funciones} / N^{\circ} \text{ invocaciones función} =$
 \Rightarrow