TEMA 1. ANÁLISIS DE PRESTACIONES EN LOS SISTEMAS COMPUTADORES

1.1. Introducción. Necesidad de las medidas estándares del rendimiento
1.2. Rendimiento
1.3. Principios cuantitativos para el diseño de computadores
1.4. Rendimiento de la CPU y del computador
1.5. Resumen de la evolución histórica del cálculo del rendimiento de un sistema computador

1.1. INTRODUCCIÓN. NECESIDAD DE LAS MEDIDAS ESTÁNDARES DEL RENDIMIENTO

La tecnología de computadores ha progresado increíblemente en los últimos setenta años. En el año 1945 aún no existía un computador comercial de programa almacenado (Arquitectura Von Neumann). Actualmente es posible comprar un computador personal por unos cuantos cientos de euros con más prestaciones, más memoria principal y más memoria de disco que un computador que, en 1965, costaba un millón de dólares. Esto es como consecuencia de los avances en las tecnologías utilizadas para su construcción, así como de las innovaciones en los diseños de los mismos. En la Figura 1.1 se muestra gráficamente el aumento de las prestaciones de los sistemas computadores entre los años 1965 y 1990.

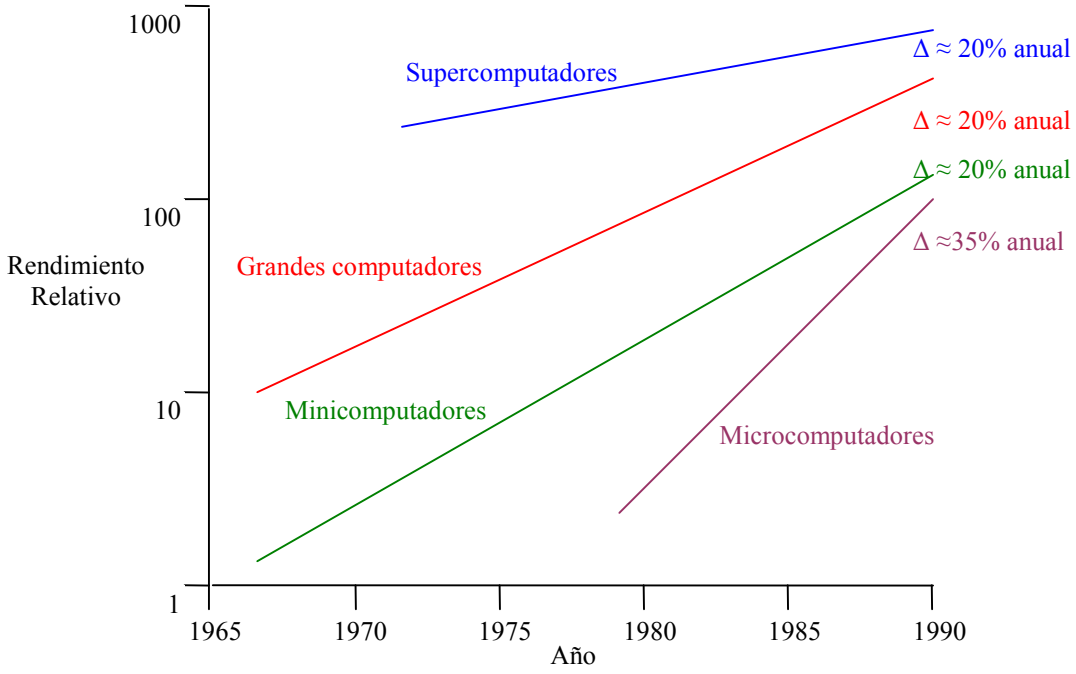

Figura 1.1. *Crecimiento del rendimiento entre los años 1965 y 1990, de diferentes clases de computadores.*
Supercomputadores → *Los más caros. Diseñados principalmente para aplicaciones científicas.*
Grandes computadores (Mainframes) → *Máquinas de propósito general de altas prestaciones.*
Minicomputadores → *Máquinas de tamaño medio.*
Microcomputadores → *Desde los computadores personales a las estaciones de trabajo.*

Mientras que las mejoras tecnológicas han sido muy constantes, el progreso en cuanto a la arquitectura ha sido bastante menos consistente. En los *grandes computadores* (*mainframes*) la velocidad de crecimiento se ha debido principalmente a la tecnología. Los *supercomputadores* han crecido gracias a las mejoras tecnológicas y arquitectónicas. En los *minicomputadores*, los avances han incluido formas innovadoras

de implementar arquitecturas y la adopción de parte de la tecnología empleada en los grandes computadores. El crecimiento del rendimiento de los *microcomputadores* ha sido el más rápido principalmente porque estas máquinas han aprovechado las ventajas de la tecnología de circuitos integrados; además, desde 1980, los microprocesadores han sido la tecnología para las nuevas arquitecturas y las nuevas implementaciones de arquitecturas antiguas.

Las innovaciones que se han producido en las arquitecturas de computadores desde mediados de los años 80, reduciendo el coste y riesgo de lanzar al mercado una nueva arquitectura, la han propiciado dos cambios significativos en el mercado:

- La eliminación virtual de la programación en ensamblador; lo que ha reducido drásticamente la necesidad de la compatibilidad del código objeto.
- La creación de sistemas operativos estandarizados, independientes de los vendedores (UNIX por ejemplo).

Aunando los avances en la tecnología de circuitos integrados, las mejoras en la tecnología de computadores y las nuevas ideas arquitectónicas, los diseñadores han podido crear una serie de máquinas que han mejorado el rendimiento en un factor de casi 2 cada año. Como consecuencia del avance tecnológico y un mayor conocimiento empírico sobre la utilización de los computadores, ha surgido un estilo de diseño de computadores basado en datos empíricos, experimentación y simulación.

1.2. RENDIMIENTO

¿Qué se quiere decir con: "*un computador es más rápido que otro*"?

- Según el *usuario*: si ejecuta un programa en menos tiempo. El usuario está interesado en reducir el *tiempo de respuesta* → tiempo transcurrido entre el comienzo y el final de un evento.
- Según el *director de un centro de cálculo*: si completa más tareas en la unidad de tiempo. El director de un centro de cálculo está interesado en incrementar la *productividad* (*throughput*) → cantidad total de trabajo realizado en la unidad de tiempo. También se le denomina a veces *ancho de banda*.

Normalmente los términos: *tiempo de respuesta*, *tiempo de ejecución* y *productividad* se utilizan cuando se está desarrollando la tarea de cálculo completa. Los términos:

latencia y *ancho de banda* casi siempre se eligen cuando se habla de un sistema de memoria. Todos estos términos los iremos empleando y definiendo.

Ejemplo 1.1. Las siguientes mejoras en el rendimiento, ¿incrementan la *productividad*, hacen disminuir el *tiempo de respuesta*, o ambas cosas?:

1. Ciclo de reloj más rápido.
2. Múltiples sistemas procesadores para tareas separadas (tratamiento del sistema de reservas de una compañía aérea, para un país por ejemplo).
3. Procesamiento paralelo de problemas científicos.

Solución Ejemplo 1.1. En 2, ninguna tarea de las que se están realizando en paralelo funciona más rápida; por tanto, sólo incrementa la productividad. Como la disminución del tiempo de respuesta, habitualmente, mejora la productividad, 1 y 3 mejoran el tiempo de respuesta y la productividad. ■

A veces todas estas medidas se describen mejor con distribuciones de probabilidad en lugar de con valores constantes. Por ejemplo, si se considera el tiempo de respuesta para completar una operación de E/S en un disco, el tiempo de respuesta depende de factores no determinísticos como: ¿qué está haciendo el disco en el instante de la petición de E/S?, ¿cuántas tareas están esperando para acceder al disco?. Como estos valores no son fijos, tiene más sentido hablar de *tiempo medio de respuesta* de un acceso al disco. De igual forma, la *productividad efectiva* del disco (número de datos que realmente va o viene del disco por unidad de tiempo), referida a dicha operación de E/S, no es un valor constante. Excepto cuando se trate la E/S, se tratarán el tiempo de respuesta y la productividad como valores determinísticos.

La comparación de dos alternativas de diseño (rendimiento de una máquina X con el rendimiento de una máquina Y) se puede hacer desde el punto de vista de tiempo de respuesta (o tiempo de ejecución) o desde el punto de vista de rendimiento (inverso del tiempo de respuesta):

- Desde el punto de vista del *tiempo de respuesta* o del *tiempo de ejecución*, la frase "*X es más rápida que Y*" se utiliza para indicar que el tiempo de respuesta o tiempo de ejecución, para una tarea dada, es inferior en la máquina X que en la máquina Y. En concreto, "*X es n% más rápida que Y*" significa:

$$tiempo\ ejecución_Y = tiempo\ ejecución_X + \frac{n}{100} \cdot tiempo\ ejecución_X \Rightarrow \frac{tiempo\ ejecución_Y}{tiempo\ ejecución_X} = 1 + \frac{n}{100}$$

Como el *tiempo de ejecución* es el inverso del *rendimiento*, se tiene la siguiente relación:

$$1 + \frac{n}{100} = \frac{tiempo\ ejecución_Y}{tiempo\ ejecución_X} = \frac{\frac{1}{rendimiento_Y}}{\frac{1}{rendimiento_X}} = \frac{rendimiento_X}{rendimiento_Y}$$

- Desde el punto de vista del *rendimiento*, un "*incremento del n% en el rendimiento de la máquina X con respecto al de la máquina Y*", se interpreta como la diferencia entre el rendimiento de la máquina más rápida (máquina X) y la más lenta (máquina Y) dividida por el rendimiento de la máquina más lenta (máquina Y); que es equivalente a la definición anterior:

$$n = \left(\frac{Rendimiento_X - Rendimiento_Y}{Rendimiento_Y} \right) \cdot 100 \Rightarrow \frac{n}{100} = \frac{Rendimiento_X}{Rendimiento_Y} - 1 \Rightarrow 1 + \frac{n}{100} = \frac{Rendimiento_X}{Rendimiento_Y}$$

Desde el punto de vista de la productividad, la frase "*la productividad de la máquina X es 30% superior que la de la máquina Y*" significa que el número de tareas por unidad de tiempo completadas en la máquina X es 1.3 veces el número de tareas completadas por unidad de tiempo en la máquina Y.

Ejemplo 1.2. Si la máquina A ejecuta un programa en diez segundos y la máquina B ejecuta el mismo programa en quince segundos, ¿cuál de las siguientes sentencias es verdadera?:

1. A es el 50% más rápida que B
2. A es el 33% más rápida que B

Solución Ejemplo 1.2. Que la máquina A sea el n% más rápida que la máquina B, significa:

$$tiempo\ ejecución_B = tiempo\ ejecución_A + \frac{n}{100} \cdot tiempo\ ejecución_A \Rightarrow \frac{tiempo\ ejecución_B}{tiempo\ ejecución_A} = 1 + \frac{n}{100} \Rightarrow$$
$$\Rightarrow \frac{n}{100} = \frac{tiempo\ ejecución_B}{tiempo\ ejecución_A} - 1 \Rightarrow n = \frac{tiempo\ ejecución_B - tiempo\ ejecución_A}{tiempo\ ejecución_A} \cdot 100 = \frac{15-10}{10} \cdot 100 = 50\%$$

Por lo tanto, la sentencia 1 es cierta. ■

Los términos *productividad* y *latencia* interactúan de forma diferente en los diseños de computadores. Por ejemplo, la segmentación (computador con paralelismo interno, a nivel de instrucción) mejora la productividad al solapar la ejecución de múltiples instrucciones pero, no mejora la latencia (incluso la puede aumentar).

1.3. PRINCIPIOS CUANTITATIVOS PARA EL DISEÑO DE COMPUTADORES

En este apartado se introducen reglas y observaciones importantes para el diseño de los computadores.

Principio: ACELERAR EL CASO COMÚN

El principio más importante y generalizado del diseño de computadores es acelerar el caso común; es decir, al realizar un diseño, favorecer el caso más frecuente sobre el menos frecuente. Este principio también se aplica a la hora de determinar cómo se emplean los recursos ya que, si se dedican los recursos para hacer más rápida una determinada tarea, cuanto más frecuente sea esta tarea más mejorará el rendimiento del sistema. Además, el caso más frecuente suele ser generalmente el más simple y, por tanto, normalmente puede realizarse de forma más rápida que el menos frecuente; por ejemplo, si se suman dos números, el que se produzca desbordamiento suele ser infrecuente, pudiéndose por tanto mejorar el rendimiento optimizando el caso más frecuente de ausencia de desbordamiento, aún cuando este hecho ralentice la situación en que se produzca desbordamiento (el rendimiento global mejora si se optimiza el caso más frecuente).

El aplicar este principio implica conocer el/los caso/s más frecuente/s y saber, una vez conocido/s, cómo influiría/n en la mejora del rendimiento al hacerlo/s más rápido/s. La cuantificación de este principio se realiza mediante la aplicación de la *Ley de Amdahl*.

La ***Ley de Amdahl*** establece que: "*la mejora obtenida en el rendimiento al utilizar algún modo de ejecución más rápido está limitada por la fracción de tiempo que se pueda utilizar ese modo más rápido*".

La *Ley de Amdahl* define la *ganancia de rendimiento* o *aceleración* (*speedup*) que puede lograrse al utilizar una característica particular. ¿Qué es la *aceleración*?:

$$aceleración\ de\ rendimiento = \frac{rendimiento\ de\ tarea\ con\ la\ mejora\left(cuando\ es\ posible\right)}{rendimiento\ de\ tarea\ sin\ la\ mejora}$$

$$aceleración\ de\ rendimiento = \frac{tiempo\ de\ ejecución\ de\ tarea\ sin\ la\ mejora}{tiempo\ de\ ejecución\ de\ tarea\ con\ la\ mejora\left(cuando\ es\ posible\right)}$$

En definitiva, la *aceleración* relaciona la rapidez con que se realizará una tarea utilizando una máquina con la mejora con la rapidez de la máquina original.

Ejemplo 1.3. Considérese el problema de viajar desde Nevada a California a través de las Montañas Nevadas y del Desierto de los Ángeles. Aunque hay disponibles varios tipos de vehículos, como el viaje se realiza a través de áreas ecológicamente sensibles (las montañas), se debe hacer a pie esas zonas, empleando veinte horas para ello. Sin embargo, existe la posibilidad de recorrer las últimas 200 millas en un vehículo. Hay cinco formas de completar la segunda parte del viaje:

1. Yendo a pie a una velocidad media de 4 millas/hora.
2. Montando en bicicleta a una velocidad media de 10 millas/hora.
3. Conducir un “Hyundai Excel” con una velocidad media de 50 millas/hora.
4. Conducir un “Ferrari Testarossa” a 120 millas/hora.
5. Conducir un vehículo oruga a una velocidad media de 600 millas/hora.

¿Cuánto se tardará en realizar el viaje completo utilizando cada uno de estos vehículos y cuál es el aumento de velocidad si se toma como referencia el recorrido a pie de la distancia completa?

Solución Ejemplo 1.3. Se puede resolver determinando cuánto dura la segunda parte del viaje y sumándoselo a las 20 horas necesarias para cruzar las montañas. La tabla 1.1 resume estos cálculos. ■

La *Ley de Amdahl* nos permite calcular la aceleración de forma rápida, aceleración que depende de dos factores:

- La fracción del tiempo de cálculo de la máquina original que puede utilizarse para aprovechar la mejora. En el ejemplo que se está considerando es:

$$Fracción_{mejorada} = \frac{50\ horas(\ parte\ sin\ mejora)}{70\ horas(total\ viaje\ sin\ mejora)} = \frac{5}{7}$$ [*Siempre menor que* 1]

- La optimización lograda por el modo de ejecución mejorado; es decir, cuánto más rápido se ejecutaría la tarea si solamente se utilizase el modo mejorado. En el ejemplo que nos ocupa, este valor aparece en la columna etiquetada como "Aceleración en el desierto" [*Siempre mayor que* 1]. A este valor le llamamos $aceleración_{mejorada}$.

El *tiempo de ejecución* utilizando la máquina mejorada será el tiempo empleado utilizando la máquina original (en la parte que no se puede mejorar) más el tiempo empleado utilizando la máquina mejorada (en la parte que se puede mejorar).

$$tiempo\ de\ ejecución_{nuevo} = tiempo\ de\ ejecución_{original} \cdot \left(\left(1 - fracción_{mejorada}\right) + \frac{fracción_{mejorada}}{aceleración_{mejorada}} \right)$$

La *aceleración global* es la relación de los *tiempos de ejecución*:

$$aceleración_{global} = \frac{tiempo\ de\ ejecución_{original}}{tiempo\ de\ ejecución_{nuevo}} = \frac{1}{\left(1 - fracción_{mejorada}\right) + \frac{fracción_{mejorada}}{aceleración_{mejorada}}}$$

Vehículo para 2ª parte del viaje	Horas 2ª parte del viaje	Aceleración en el desierto	Horas de viaje completo	Aceleración en viaje completo
A pie	50.00	1.00	70.00	1.00
Bicicleta	20.00	2.50	40.00	1.75
Hyundai Excel	4.00	12.50	24.00	2.92
Testarossa	1.67	29.94	21.67	3.23
Vehículo oruga	0.33	151.52	20.33	3.44

Tabla 1.1. *Aceleraciones obtenidas para los diferentes medios de transporte.*

Ejemplo 1.4. Considérese una mejora que haga que la máquina corra 10 veces más rápida que la original, pero sólo es utilizable el 40% del tiempo. ¿Cuál es la aceleración global lograda al incorporar la mejora?

Solución Ejemplo 1.4. Se tiene:

$$fracción_{mejorada} = 0.4$$

$$aceleración_{mejorada} = 10$$

Por lo tanto:

$$aceleración_{global} = \frac{tiempo\ de\ ejecución_{original}}{tiempo\ de\ ejecución_{nuevo}} = \frac{1}{\left(1 - fracción_{mejorada}\right) + \frac{fracción_{mejorada}}{aceleración_{mejorada}}} =$$

$$= \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} = 1.5625$$

Para este ejemplo $aceleración_{global}$ = 1.5625. ■

La *Ley de Amdahl* expresa la *ley de rendimientos decrecientes*: "*la mejora incremental en la aceleración conseguida por una mejora adicional en el rendimiento de una parte de cálculo disminuye tal como se van añadiendo mejoras*".

Un *corolario* importante de la la *Ley de Amdahl* es que: "*si una mejora sólo es utilizable por una fracción de una tarea, no podemos aumentar la velocidad de la tarea más que el inverso de 1 menos esa fracción*".

Un error común al aplicar la *Ley de Amdahl* es confundir "*fracción de tiempo convertido para utilizar una mejora*" y "*fracción de tiempo después de que se utiliza la mejora*". Si en lugar de medir el tiempo que **podría utilizar** la mejora en un cálculo, se mide el tiempo **después** de que se ha utilizado la mejora, los resultados serían incorrectos.

La Ley de Amdahl puede servir como guía para ver cómo una mejora aumenta el rendimiento y cómo distribuir los recursos para mejorar la relación coste/rendimiento. El

objetivo claramente es *emplear recursos de forma proporcional al tiempo que se requiere en cada parte*.

Ejemplo 1.5. Supongamos que se quiere mejorar la velocidad de la CPU de nuestra máquina en un factor de 5 (sin afectar al rendimiento de E/S) por 5 veces el coste. Supongamos también que la CPU se utiliza el 50% del tiempo, y que el tiempo restante la CPU está esperando las E/S. Si la CPU supone 1/3 del coste total del computador, ¿el incremento de la velocidad de la CPU en un factor de 5 es una buena inversión desde un punto de vista coste/rendimiento?.

Solución Ejemplo 1.5. La aceleración obtenida es:

$$aceleración = \frac{1}{0.5 + \frac{0.5}{5}} = \frac{1}{0.6} = 1.67$$

La nueva máquina costaría:

$$coste\ máquina\ nueva = \left(\frac{2}{3}\cdot 1 + \frac{1}{3}\cdot 5\right)\cdot coste\ de\ la\ máquina\ original =$$

$$= 2.33\ veces\ el\ coste\ de\ la\ máquina\ original$$

Como el incremento de coste es mayor que la mejora de rendimiento, este cambio no mejora la relación coste/rendimiento. ■

Ejemplo 1.6. Se necesita mejorar el código de cierta aplicación y se tienen dos posibilidades:

1. Acelerar en un factor de 10 uno de los módulos del programa que supone un 10% del tiempo de ejecución.
2. Acelerar un factor 2 uno de los módulos del programa que supone un 50% del tiempo de ejecución.

¿Cuál es la mejor opción?

Solución Ejemplo 1.6. Las aceleraciones correspondientes son:

$$aceleración_1 = \frac{1}{0.9 + \frac{0.1}{10}} = \frac{1}{0.91} = 1.10$$

$$aceleración_2 = \frac{1}{0.5 + \frac{0.5}{2}} = \frac{1}{0.75} = 1.33$$

Por lo tanto, la opción 2 es preferible a la opción 1. ■

Ejemplo 1.7. Se quiere mejorar el rendimiento de cierto computador para cierta aplicación y se tienen las siguientes opciones:

1. Cambiar el procesador gráfico que se utiliza un 20% del tiempo, consiguiendo una aceleración de 20.
2. Incrementar la memoria obteniendo una aceleración de 1.5 el 80% del tiempo.

¿Cuál es la mejor opción?

Solución Ejemplo 1.7. Las aceleraciones correspondientes son:

$$aceleración_1 = \frac{1}{0.8 + \frac{0.2}{20}} = \frac{1}{0.81} = 1.23$$

$$aceleración_2 = \frac{1}{0.2 + \frac{0.8}{1.5}} = \frac{1}{0.73} = 1.37$$

Por lo tanto, la opción 2 es preferible a la opción 1. ■

Propiedad de los programas: LOCALIDAD DE REFERENCIA

Aunque la Ley de Amdahl es un teorema que se aplica a cualquier sistema, otras observaciones importantes provienen de las propiedades de los programas. La propiedad más importante, que se explota regularmente de un programa, es la *localidad de referencia*: "*los programas tienden a reutilizar los datos e instrucciones que han utilizado recientemente*". Una regla empírica muy corroborada es que: "*un programa emplea el 90% de su tiempo de ejecución en sólo el 10% del código*". Gracias a la localidad de referencia de los programas, se puede predecir con una precisión razonable, basándonos

en el pasado reciente del programa, qué instrucciones y datos utilizará un programa en el futuro próximo.

La localidad de referencia se aplica tanto a los accesos a las instrucciones como a los accesos a los datos. Se han observado dos tipos diferentes de *localidad de referencia*:

- Localidad temporal: especifica que "*los elementos accedidos recientemente, probablemente serán accedidos en un futuro*".
- Localidad espacial: establece que "*los elementos cuyas direcciones son próximas tienden a ser referenciados juntos en el tiempo*".

1.4. RENDIMIENTO DE LA CPU Y DEL COMPUTADOR

Nos planteamos las siguientes preguntas: ¿Por qué los ingenieros diseñan computadores diferentes? ¿Cómo los clientes se deciden por un computador en vez de por otro? ¿Hay una base racional para estas decisiones?. Si es así, ¿pueden los ingenieros utilizar esta base para diseñar mejores computadores?

Las medidas estándares del rendimiento proporcionan una base para realizar una comparación y saber cómo mejorar el objeto medido. Estas medidas permiten a los diseñadores realizar una selección cuantitativa entre distintas alternativas, lográndose un progreso ordenado en el campo de los computadores.

El rendimiento y el coste forman la base racional para decidir qué computador elegir. Por lo tanto, el arquitecto de computador debe tener en cuenta estos dos parámetros a la hora de diseñar un computador. El diseñador se encuentra ante un dilema, aumentar el rendimiento o reducir el coste. En un extremo está el *diseño de alto rendimiento*, que no escatima en costes para conseguir sus objetivos (computadores Cray por ejemplo). En el otro extremo está el *diseño de bajo coste*, que sacrifica rendimiento para conseguir un coste más bajo (computadores tipo PC). Entre estos extremos está el *diseño coste/rendimiento*, en el que el diseñador equilibra el coste frente a rendimiento (estaciones de trabajo).

Normalmente se habla de rendimientos en términos de tiempo o de velocidad: "*El computador que realiza la misma cantidad de trabajo en el mínimo tiempo es el más rápido*"; "*El tiempo de ejecución de un programa es el tiempo, medido en segundos, que tarda en ejecutarse*". Puesto que el tiempo más bajo significa mayor rendimiento, "*El rendimiento se mide como una frecuencia de eventos por segundo*".

El tiempo se puede definir de formas distintas dependiendo de lo que queramos medir. La definición más directa de tiempo es: *tiempo de reloj*, *tiempo de respuesta* o *tiempo transcurrido* (*elapsed time*): *tiempo empleado para completar una tarea* (incluyendo accesos a disco, accesos a memoria, actividades de E/S, gastos del SO; todo. Sin embargo, si se tiene multiprogramación, una CPU puede trabajar sobre otro programa mientras espera la E/S y, como consecuencia, no minimizar el tiempo transcurrido de un programa. Para este caso, es necesario algún término que tenga en cuenta esta actividad; el *tiempo de CPU*: "*tiempo que la CPU está calculando, sin incluir el tiempo de espera para la E/S o para ejecutar otros programas*". Evidentemente, el tiempo de respuesta que percibe el usuario es el *tiempo transcurrido* del programa y no el *tiempo de CPU*. Además, el *tiempo de CPU* puede dividirse en *tiempo de CPU del usuario* (tiempo empleado por la CPU en el programa) y *tiempo de CPU del sistema* (tiempo empleado por el SO realizando tareas requeridas por el programa):

$$tiempo\ de\ CPU = tiempo\ de\ CPU\ del\ usuario + tiempo\ de\ CPU\ del\ sistema$$

Por ejemplo,

$$\left.\begin{array}{l} tiempo\ transcurrido = 159\ segundos \\ tiempo\ de\ CPU\ del\ usuario = 90.7\ segundos \\ tiempo\ de\ CPU\ del\ sistema = 12.9\ segundos \end{array}\right\} \Rightarrow tiempo\ de\ CPU = 103.6\ segundos \Bigg\} \Rightarrow$$

$$\Rightarrow tiempo\ en\ E/S\ y\ otros\ programas = 55.4\ segundos$$

En el ejemplo planteado, más de $1/3$ del tiempo ha transcurrido esperando las E/S o ejecutando otros programas, o ambas cosas.

Existe una discusión sobre si incluir o no el *tiempo del sistema* en el *tiempo de CPU*. Aunque ambas partes tienen razón, el criterio que se va a seguir a partir de ahora es:

$$rendimiento\ del\ sistema = tiempo\ transcurrido\ en\ un\ sistema\ no\ cargado$$

$$rendimiento\ de\ la\ CPU = tiempo\ de\ CPU\ del\ usuario$$

La mayoría de los computadores se diseñan como sistemas síncronos, que utilizan una señal de reloj funcionando a una frecuencia constante; cada uno de estos periodos se denomina *ciclo* o *ciclo de reloj*. Los diseñadores de computadores hacen referencia a

estos ciclos por su duración (por ejemplo, 10 ns) o por su frecuencia (100 MHz). Por lo tanto, se puede expresar el *tiempo de CPU* para un programa de dos formas:

$$tiempo\ de\ CPU = Ciclos\ de\ reloj\ de\ CPU \cdot tiempo\ de\ Ciclo$$

$$tiempo\ de\ CPU = \frac{Ciclos\ de\ reloj\ de\ CPU}{Frecuencia\ de\ reloj}$$

Se observa que no tendría sentido mostrar el *tiempo transcurrido* en función de la duración del ciclo de reloj, ya que la latencia de los dispositivos de E/S, normalmente, es independiente de la frecuencia de reloj de la CPU.

Si por *recuento de instrucciones* se entiende el número de instrucciones ejecutadas, el número medio de *ciclos de reloj por instrucción* (*CPI*) es igual a:

$$CPI = \frac{Ciclos\ de\ reloj\ de\ CPU\ \ para\ un\ programa}{Recuento\ de\ instrucciones}$$

Considerando este término, se tiene:

$$tiempo\ de\ CPU = Ciclos\ de\ reloj\ de\ CPU \cdot tiempo\ de\ Ciclo = Recuento\ de\ instrucciones \cdot CPI \cdot tiempo\ de\ Ciclo$$

$$tiempo\ de\ CPU = \frac{Ciclos\ de\ reloj\ de\ CPU}{Frecuencia\ de\ reloj} = \frac{Recuento\ de\ instrucciones \cdot CPI}{Frecuencia\ de\ reloj}$$

Por lo tanto:

$$tiempo\ de\ CPU = \frac{Segundos}{Programa} = \frac{Instrucciones}{Programa} \cdot \frac{Ciclos\ de\ reloj}{Instrucción} \cdot \frac{Segundos}{Ciclo\ de\ reloj}$$

Es decir, el rendimiento de la CPU depende de tres características:

- Ciclo de reloj (o frecuencia)
- Ciclos de reloj por instrucción
- Recuento de instrucciones

Ninguna de las características anteriores se pueden cambiar sin tener en cuenta las demás:

- La frecuencia de reloj depende de la tecnología hardware y de la organización.
- El CPI depende de la organización y de la arquitectura a nivel de lenguaje máquina.
- El recuento de instrucciones depende de la arquitectura a nivel de lenguaje máquina y de la tecnología de los compiladores.

A veces interesa ser más exacto en el cálculo del número total de ciclos de reloj de la CPU, realizando el mismo como:

$$Ciclos\ de\ reloj\ de\ CPU = \sum_{i=1}^{n} \left(CPI_i \cdot I_i \right)$$

donde I_i representa el número de veces que se ejecuta la instrucción i en un programa y CPI_i representa el número medio de ciclos de reloj para la instrucción i. Quedando el tiempo de CPU como:

$$tiempo\ de\ CPU = tiempo\ de\ Ciclo \cdot \sum_{i=1}^{n} \left(CPI_i \cdot I_i \right)$$

El *CPI global* queda como:

$$CPI = \frac{\sum_{i=1}^{n} \left(CPI_i \cdot I_i \right)}{Recuento\ de\ instrucciones} = \sum_{i=1}^{n} \left(CPI_i \cdot \frac{I_i}{Recuento\ de\ instrucciones} \right)$$

$$Fracción\ de\ ocurrencia\ de\ una\ instrucción\ en\ un\ programa = \frac{I_i}{Recuento\ de\ instrucciones}$$

El parámetro CPI_i debe medirse y no calcularse a partir de una tabla al final del manual de referencia, ya que debe incluir los fallos de caché y el resto de ineficiencias del sistema de memoria.

Es necesario tener siempre en cuenta que la medida real del rendimiento del computador es el tiempo. Si se cambia el repertorio de instrucciones del computador, por ejemplo puede conducir a una organización con un ciclo de reloj de mayor duración, que contrarrestaría las mejoras obtenidas con el recuento de instrucciones. Por lo tanto:

"*cuando se comparan dos máquinas se deben examinar los tres componentes (frecuencia de reloj, CPI y recuento de instrucciones) para comprender el rendimiento relativo*"

Ejemplo 1.8. Suponer que se están considerando dos alternativas para una instrucción de salto condicional:

CPU A. Una instrucción de comparación define el valor de los biestables del registro de estado, y es seguida por una instrucción de salto que examina el código de condición.

CPU B. La comparación se incluye en la instrucción de salto.

Suponer que en ambas CPUs, la instrucción de salto condicional emplea dos ciclos de reloj, y el resto de instrucciones un ciclo. (Es obvio que si el CPI es 1.0, excepto en los saltos, se está ignorando las pérdidas debidas al sistema de memoria). En la CPU A, el 20% de todas las instrucciones ejecutadas son saltos condicionales; como cada salto necesita una comparación previa, otro 20% de las instrucciones son comparaciones. Debido a que la CPU A no incluye la comparación en el salto, su ciclo de reloj es un 25% más rápido que el de la CPU B. ¿Qué CPU es más rápida?

Solución Ejemplo 1.8. Como se ignoran todas las prestaciones del sistema de memoria, se puede utilizar la fórmula del rendimiento de la CPU:

$$tiempo\ de\ CPU = Ciclos\ de\ reloj\ de\ CPU \cdot tiempo\ de\ Ciclo = Recuento\ de\ instrucciones \cdot CPI \cdot tiempo\ de\ Ciclo$$

En la CPU A, los ciclos por instrucción son: $CPI_A = (0.20 \cdot 2) + (0.80 \cdot 1) = 1.2$, ya que el 20% de las instrucciones son de salto condicionales (de duración 2 ciclos de reloj) y, el 80% restante tienen una duración de 1 ciclo de reloj.

En la CPU B, los ciclos de reloj por instrucción son: $CPI_B = (0.25 \cdot 2) + (0.75 \cdot 1) = 1.25$, ya que el 20% de 80 instrucciones (no hay instrucciones de comparación) es 0.25 (de duración 2 ciclos de reloj) y, por lo tanto, el 75% restante tiene una duración de 1 ciclo de reloj.

La duración del ciclo de reloj de la CPU B es: $Ciclo_{CPU\ B} = 1.25 \cdot Ciclo_{CPU\ A}$ ya que el $Ciclo_{CPU\ A}$ es el 25% más rápido que el $Ciclo_{CPU\ B}$.

Los tiempos transcurridos, en función de la duración del ciclo de reloj correspondiente, de las dos CPUs son:

$$\begin{aligned} tiempo\ de\ CPU_A &= Recuento\ de\ instrucciones_A \cdot CPI_A \cdot tiempo\ de\ Ciclo_A = \\ &= Recuento\ de\ instrucciones_A \cdot 1.2 \cdot tiempo\ de\ Ciclo_A = \\ &= 1.2 \cdot Recuento\ de\ instrucciones_A \cdot tiempo\ de\ Ciclo_A \end{aligned}$$

$$\begin{aligned} tiempo\ de\ CPU_B &= Recuento\ de\ instrucciones_B \cdot CPI_B \cdot tiempo\ de\ Ciclo_B = \\ &= \left(0.80 \cdot Recuento\ de\ instrucciones_A\right) \cdot 1.25 \cdot \left(1.25 \cdot tiempo\ de\ Ciclo_A\right) = \\ &= 1.25 \cdot Recuento\ de\ instrucciones_A \cdot tiempo\ de\ Ciclo_A \end{aligned}$$

Por lo tanto, la CPU A, con un ciclo de reloj más corto, es más rápida que la CPU B, que ejecuta un número menor de instrucciones. ■

Ejemplo 1.9. Después del análisis de los resultados del ejemplo anterior, un diseñador consideró que modificando la organización del computador, las diferencias de las duraciones de los ciclos de reloj de ambas máquinas podrían reducirse a un 10%. ¿Qué CPU es más rápida ahora?

Solución Ejemplo 1.9. El único cambio ahora con respecto a los datos del ejemplo anterior es que $Ciclo_{CPU\ B} = 1.10 \cdot Ciclo_{CPU\ A}$. Por lo tanto:

$$tiempo\ de\ CPU_A = 1.2 \cdot Recuento\ de\ instrucciones_A \cdot tiempo\ de\ Ciclo_A$$

$$\begin{aligned} tiempo\ de\ CPU_B &= Recuento\ de\ instrucciones_B \cdot CPI_B \cdot tiempo\ de\ Ciclo_B = \\ &= \left(0.80 \cdot Recuento\ de\ instrucciones_A\right) \cdot 1.25 \cdot \left(1.10 \cdot tiempo\ de\ Ciclo_A\right) = \\ &= 1.10 \cdot Recuento\ de\ instrucciones_A \cdot tiempo\ de\ Ciclo_A \end{aligned}$$

Con la mejora propuesta, resulta ahora más rápida la CPU B que la CPU A, ejecutando menos instrucciones. ■

Ejemplo 1.10. Supongamos que se está considerando un cambio en un repertorio de instrucciones. El computador, inicialmente sólo tiene instrucciones de carga y de almacenamiento en memoria, realizándose después todas las operaciones en registros. Tales computadores se denominan computadores de *carga/almacenamiento* (*load/store*). En la tabla 1.2 se dan las medidas del computador o máquina de carga/almacenamiento que muestran la frecuencia de instrucciones, denominada *mezcla de instrucciones* (*instruction mix*), y el número de ciclos de reloj por instrucción.

Supóngase que el 25% de las operaciones que realiza la ALU utilizan directamente un operando cargado que no se utiliza de nuevo.

Operación	Frecuencia	Ciclos de reloj/Instrucción
Operaciones ALU	43 %	1
Cargas	21 %	2
Almacenamientos	12 %	2
Saltos	24 %	2

Tabla 1.2. *Ejemplo de frecuencia de instrucciones.*

Se propone añadir instrucciones a la ALU que tengan un operando fuente en memoria. Estas nuevas *instrucciones de registro - memoria* emplean 2 ciclos de reloj. Supongamos que el repertorio extendido de instrucciones incrementa en 1 el número de ciclos de reloj para los saltos, sin afectar a la duración del ciclo de reloj (ya se explicará más adelante por qué las instrucciones de registro – memoria pueden ralentizar los saltos). ¿Mejorará este cambio el rendimiento de la CPU?

Solución Ejemplo 1.10. La pregunta en cuestión, es si la nueva máquina es más rápida que la original. Se va a emplear la fórmula que nos da el rendimiento de la CPU, puesto que se está ignorando las prestaciones del sistema:

$$tiempo\ de\ CPU = Recuento\ de\ instrucciones \cdot CPI \cdot tiempo\ de\ Ciclo$$

Según la tabla 1.2, el CPI la CPU original es:

$$CPI_{Original} = \left(0.43 \cdot 1 + 0.21 \cdot 2 + 0.12 \cdot 2 + 0.24 \cdot 2\right) = 1.57$$

Por lo tanto, el rendimiento de la $CPU_{Original}$ es:

$$\begin{aligned} tiempo\ de\ CPU_{Original} &= Recuento\ de\ instrucciones_{Original} \cdot CPI_{Original} \cdot tiempo\ de\ Ciclo_{Original} = \\ &= 1.57 \cdot Recuento\ de\ instrucciones_{Original} \cdot tiempo\ de\ Ciclo_{Original} \end{aligned}$$

El CPI de la máquina nueva es:

$$CPI_{Nueva} = \frac{(0.43 - 0.25 \cdot 0.43) \cdot 1 + (0.21 - (0.25 \cdot 0.43)).2 + (0.25 \cdot 0.43) \cdot 2 + 0.12 \cdot 2 + 0.24 \cdot 3}{1 - (0.25 \cdot 0.43)}$$

Para calcular el CPI de la máquina nueva se ha tenido en cuenta que el 25% de las instrucciones ALU (que son el 43% de todas las instrucciones ejecutadas) pasan a ser instrucciones registro – memoria, que afecta a las tres primeras componentes del numerador: hay $(0.25 \cdot 0.43)$ operaciones ALU menos, $(0.25 \cdot 0.43)$ operaciones de carga menos, y $(0.25 \cdot 0.43)$ nuevas instrucciones registro – memoria de la ALU. El resto del numerador permanece igual, excepto los saltos que ahora emplean 3 ciclos de reloj en lugar de 2. Por último, se divide por el nuevo recuento de instrucciones que es $(0.25 \cdot 0.43)$ menos que el original. Por lo tanto, el resultado final es:

$$CPI_{Nueva} = \frac{1.7025}{0.8925} = 1.908$$

Como la duración del ciclo de reloj es inalterable, el rendimiento de la CPU es:

$$\begin{aligned} tiempo\ de\ CPU_{Nueva} &= Recuento\ de\ instrucciones_{Nueva} \cdot CPI_{Nueva} \cdot tiempo\ de\ Ciclo_{Nueva} = \\ &= (0.8925 \cdot Recuento\ de\ instrucciones_{Original}) \cdot 1.908 \cdot tiempo\ de\ Ciclo_{Original} = \\ &= 1.7029 \cdot Recuento\ de\ instrucciones_{Original} \cdot tiempo\ de\ Ciclo_{Original} \end{aligned}$$

La conclusión a la que se llega es que no es una buena idea añadir instrucciones registro – memoria, porque no contrarrestan el aumento del tiempo de ejecución debido a los saltos que serían más lentos en la nueva máquina. ■

MIPS Y ERRORES DE UTILIZACIÓN

Buscando una medida estándar del rendimiento de los computadores, se han adoptado una serie de medidas populares que posteriormente se han usado para algo que nunca se imaginó. La única medida fiable y consistente del rendimiento es el tiempo de ejecución de los programas reales; las demás alternativas al tiempo como métrica o a los programas reales como <<items>> medidos, han conducido a afirmaciones erróneas e incluso a errores en el diseño de computadores. Veamos algunos errores.

Una alternativa al tiempo como métrica son los MIPS (*Millones de Instrucciones Por Segundo*). Por lo tanto, para un programa dado:

$$MIPS = \frac{Recuento\ de\ instrucciones}{Tiempo\ de\ ejecución \cdot 10^6}$$

Si sustituimos en esa fórmula el tiempo de ejecución como función del recuento de instrucciones, del CPI y del tiempo de ciclo, y como la frecuencia de reloj es la inversa del tiempo de ciclo:

$$tiempo\ de\ CPU = Tiempo\ de\ ejecución = Recuento\ de\ instrucciones \cdot CPI \cdot tiempo\ de\ Ciclo$$

$$tiempo\ de\ ciclo = \frac{1}{Frecuencia\ de\ reloj}$$

se tiene:

$$MIPS = \frac{Recuento\ de\ instrucciones}{Tiempo\ de\ ejecución \cdot 10^6} = \frac{Recuento\ de\ instrucciones}{Recuento\ de\ instrucciones \cdot CPI \cdot tiempo\ de\ Ciclo \cdot 10^6} =$$

$$= \frac{Frecuencia\ de\ reloj}{CPI \cdot 10^6}$$

Por lo tanto, una segunda fórmula para los MIPS es:

$$MIPS = \frac{Frecuencia\ de\ reloj}{CPI \cdot 10^6}$$

Despejando de la fórmula $MIPS = \frac{Recuento\ de\ instrucciones}{Tiempo\ de\ ejecución \cdot 10^6}$ el Tiempo de ejecución, queda:

$$Tiempo\ de\ ejecución = \frac{Recuento\ de\ instrucciones}{MIPS \cdot 10^6}$$

y, como el rendimiento es la inversa del tiempo de ejecución, las máquinas más rápidas tendrán una mayor frecuencia de MIPS.

Ventajas de emplear MIPS para medir el rendimiento:

- Son fáciles de comprender para los clientes ya que máquinas más rápidas significan un mayor número de MIPS, que coincide con la intuición.

Inconvenientes de emplear MIPS para comparar distintas máquinas:

- Los MIPS son dependientes del repertorio de instrucciones; haciendo difícil la comparación de los MIPS de computadores con repertorios de instrucciones diferentes.
- Los MIPS varían entre programas en el mismo computador; dependen de la duración de las instrucciones que se empleen en ellos.
- Los MIPS pueden variar inversamente al rendimiento. Por ejemplo, la variación de los MIPS en una máquina con hardware opcional de coma flotante: como generalmente se emplean más ciclos de reloj por instrucción en coma flotante que por una instrucción entera, los programas en coma flotante que utilizan el hardware opcional, en lugar de las rutinas software de coma flotante, emplean menos tiempo en ejecutarse pero los MIPS obtenidos son menores (las rutinas software para resolver coma flotante emplean instrucciones simples, dando como resultado una mayor frecuencia de MIPS, pero se ejecutan tantas veces que el tiempo de ejecución es mayor que si hiciéramos las operaciones con un procesador de coma flotante).

Ejemplo 1.11. (Demuestra las anomalías al emplear MIPS como medida en compiladores optimizadores). Supongamos que se construye un compilador optimizado para la máquina de carga/almacenamiento descrita en los ejemplos 1.9 y 1.10 anteriores. El compilador descarta el 50% de las instrucciones de la ALU aunque no pueda reducir ni cargas, ni almacenamientos, ni saltos. Ignorando las prestaciones del sistema y suponiendo una duración del ciclo de reloj de 20 ns (frecuencia de reloj 50 MHz), ¿cuál es la frecuencia en MIPS para el código optimizado frente al código sin optimizar?; ¿está el <<ranking>> de los MIPS de acuerdo con el <<ranking>> del tiempo de ejecución?

Solución Ejemplo 1.11. Según se ha visto en los ejemplos anteriores:

$$CPI_{Código\ original} = \left(0.43 \cdot 1 + 0.21 \cdot 2 + 0.12 \cdot 2 + 0.24 \cdot 2\right) = 1.57$$

por tanto:

$$MIPS_{Código\ original} = \frac{Frecuencia\ de\ reloj}{CPI_{Código\ Original} \cdot 10^6} = \frac{50\ MHz}{1.57 \cdot 10^6} = \frac{50 \cdot 10^6\ Hz}{1.57 \cdot 10^6} = 31.85$$

El rendimiento del código sin optimizar es:

$$\begin{aligned} tiempo\ de\ CPU_{Código\ original} &= Recuento\ de\ instrucciones_{Código\ original} \cdot CPI_{Código\ original} \cdot tiempo\ de\ Ciclo = \\ &= Recuento\ de\ instrucciones_{Código\ original} \cdot 1.57 \cdot tiempo\ de\ Ciclo \\ &= 1.57 \cdot \left(20 \cdot 10^{-9}\right) \cdot Recuento\ de\ instrucciones_{Código\ original} = \\ &= 31.4 \cdot 10^{-9} \cdot Recuento\ de\ instrucciones_{Código\ original} \end{aligned}$$

Para el código optimizado se tendrá:

$$CPI_{Código\ Optimizado} = \frac{\left(0.43/2\right) \cdot 1 + 0.21 \cdot 2 + 0.12 \cdot 2 + 0.24 \cdot 2}{1 - \left(0.43/2\right)} = \frac{1.355}{0.785} = 1.73$$

ya que la mitad de las instrucciones de la ALU están descartadas (0.43/2) y la cuenta de instrucciones se reduce por las instrucciones que faltan de la ALU. Para este código optimizado, la frecuencia de MIPS será:

$$MIPS_{Código\ Optimizado} = \frac{Frecuencia\ de\ reloj}{CPI_{Código\ Optimizado} \cdot 10^6} = \frac{50\ MHz}{1.73 \cdot 10^6} = \frac{50 \cdot 10^6\ Hz}{1.73 \cdot 10^6} = 28.90$$

El rendimiento del código optimizado es:

$$\begin{aligned} tiempo\ de\ CPU_{Código\ Optimizado} &= Recuento\ de\ instrucciones_{Código\ Optimizado} \cdot CPI_{Código\ Optimizado} \cdot tiempo\ de\ Ciclo = \\ &= 0.785 \cdot Recuento\ de\ instrucciones_{Código\ original} \cdot 1.73 \cdot \left(20 \cdot 10^{-9}\right) = \\ &= 27.16 \cdot 10^{-9} \cdot Recuento\ de\ instrucciones_{Código\ original} \end{aligned}$$

Comparando los tiempos de CPU:

$$1 + \frac{n}{100} = \frac{tiempo\ ejecución_{Código\ original}}{tiempo\ ejecución_{Código\ Optimizado}} = \frac{31.4 \cdot 10^{-9} \cdot Recuento\ de\ instrucciones_{Código\ original}}{27.16 \cdot 10^{-9} \cdot Recuento\ de\ instrucciones_{Código\ original}} = 1.156 \Rightarrow$$

$$\Rightarrow \frac{n}{100} = 0.156$$

El código optimizado es el 15.6% más rápido que el código original, pero su frecuencia en MIPS es inferior. ■

El ejemplo 1.11 anterior muestra que los MIPS pueden fallar al dar una visión verdadera del rendimiento; por no reflejar el tiempo de ejecución. Para compensar esta carencia, otra alternativa al tiempo de ejecución es utilizar una máquina particular como punto de referencia, con una estimación convenida sobre sus MIPS. Los $MIPS_{Relativos}$, para distinguirlos de la forma original (denominados $MIPS_{Nativos}$) se calculan como se muestra a continuación:

$$MIPS_{Relativos} = \frac{Tiempo_{Referencia}}{Tiempo_{No\ estimado}} \cdot MIPS_{Referencia}$$

donde

$Tiempo_{Referencia}$ = Tiempo de ejecución de un programa en la máquina de referencia

$Tiempo_{No\ estimado}$ = Tiempo de ejecución del mismo programa en la máquina que se va a medir

$MIPS_{Referencia}$ = Estimación convenida sobre los MIPS de la máquina de referencia

Los MIPS relativos tienen en cuenta el tiempo de ejecución para el programa y la entrada de dados. Aún estando identificadas la máquina y los programas a correr en ella, según va pasando el tiempo, cada vez resulta más difícil encontrar una máquina de referencia en la que ejecutar los nuevos programas. (En los años 80, la máquina de referencia dominante era el sistema VAX-11/780, que se denominó máquina 1-MIPS). Pero, nos preguntamos si en la máquina más antigua deben correr las ediciones más modernas del compilador y sistema operativo, o si el software debe fijarse para que la máquina de referencia no sea más rápida a lo largo del tiempo.

También existe la tentación de generalizar los MIPS relativos utilizando un benchmark a tiempo de ejecución relativo, aún cuando haya amplias variaciones en el rendimiento relativo.

En resumen, la ventaja de los MIPS relativos es pequeña ya que el tiempo de ejecución, el programa y la entrada del programa deben ser conocidos, para que tengan información significativa.

MFLOPS Y ERRORES DE UTILIZACIÓN

Otra alternativa popular al tiempo de ejecución son los *millones de operaciones en punto flotante* (MFLOPS - megaflops). Según la propia definición:

$$MFLOPS = \frac{Número\ de\ operaciones\ de\ punto\ flotante\ de\ un\ programa}{Tiempo\ de\ ejecución \cdot 10^6}$$

Resulta evidente que una estimación en MFLOPS depende de la máquina y del programa. Como los MFLOPS se pensaron para medir el rendimiento en punto flotante, no son aplicables fuera de ese rango. Los compiladores, como caso extremo, tienen una estimación de MFLOPS próxima a cero (raramente utilizan aritmética en punto flotante) sin que importe lo rápido que sea la máquina.

El término MFLOPS está basado en las operaciones en lugar de en las instrucciones. Se pensó para que fuera una comparación buena entre diferentes máquinas: se creyó que el mismo programa corriendo en máquinas diferentes debía ejecutar un número diferente de instrucciones , pero el mismo número de operaciones en punto flotante. Desgraciadamente, los MFLOPS no son fiables porque:

- El conjunto de operaciones en punto flotante no es consistente con las máquinas. Por ejemplo, el CRAY-2 no tiene instrucciones de dividir, mientras que el Motorola 68882 tiene división, raíz cuadrada, seno y coseno.
- La estimación en MFLOPS cambia no sólo en la mezcla de operaciones de enteros y punto flotante, sino también en la mezcla de operaciones rápidas y lentas de punto flotante. Por ejemplo, un programa con el 100% de sumas en punto flotante tendrá una estimación en MFLOPS mayor que un programa con el 100% de divisiones en punto flotante.

Operaciones reales en PF	**Operaciones normalizadas en PF**
ADD, SUB, COMPARE, MULT	1
DIVIDE, SQRT	4
EXP, SIN, …	8

Tabla 1.3. *Operaciones de Punto Flotante reales frente a las normalizadas. Número de operaciones normalizadas en punto flotante por operación real en un programa utilizado por los autores del <<Livermore FORTRAN Kernels>> o <<Livermore Loops>>, para calcular los MFLOPS. Un núcleo (Kernel) con una suma*

(ADD), una división (DIVIDE), y un seno (SIN) necesitaría 13 operaciones normalizadas en punto flotante. Los MFLOPS nativos no darán los resultados obtenidos para otras máquinas con ese benchmark.

La solución para ambos problemas es dar un número canónico de operaciones de punto flotante a nivel de programa fuente y, después, dividirlo por el tiempo de ejecución. La tabla 1.3 muestra que en el benchmark <<Livermore Loops>> sus autores calculan el *número de operaciones normalizadas en punto flotante por programa* de acuerdo con las operaciones encontradas realmente en el código fuente. Por tanto, la estimación en *MFLOPS nativos* no es la misma que la de *MFLOPS normalizados* citados en la literatura de supercomputadores, lo cual ha sido una sorpresa para algunos diseñadores de computadores.

Ejemplo 1.12. El programa Spice se ejecuta en la DECstation 3100 en 94 segundos. El número de operaciones de punto flotante ejecutadas en ese programa se indica en la tabla 1.4. ¿Cuántos son los MFLOPS nativos para ese programa?. Usando las conversiones de la tabla 1.3, ¿cuántos son los MFLOPS normalizados?

Operación	**Nº de operaciones**
ADDD	25 999 440
SUBD	18 266 439
MULD	33 880 810
DIVD	15 682 333
COMPARED	9 745 930
NEGD	2 617 846
ABSD	2 195 930
CONVERTD	1 581 450
TOTAL	**109 970 178**

Tabla 1.4. *Número de operaciones y tipos del programa Spice ejecutándose en un computador DECstation 3100.*

Solución Ejemplo 1.12. Los MFLOPS nativos se calculan fácilmente:

$$MFLOPS_{nativos} = \frac{N^o\ de\ operaciones\ de\ punto\ flotante\ de\ un\ programa}{Tiempo\ de\ ejecución \cdot 10^6} \approx \frac{110\,M}{94 \cdot 10^6} \approx 1.2$$

La única operación que según la tabla 1.3 cambia para los MFLOPS normalizados y está en la tabla 1.4 es la de DIVIDE (4 operaciones normalizadas); teniendo en cuenta esta normalización, los MFLOPS normalizados son:

$$MFLOPS_{normalizados} \approx \frac{157\,M}{94 \cdot 10^6} \approx 1.7$$

Por lo tanto, los MFLOPS han pasado de 1.2 a 1.7. ■

Como cualquier otra medida de rendimiento, la estimación en MFLOPS para un programa no puede generalizarse para establecer una única métrica de rendimiento para un computador. Como los MFLOPS normalizados representan realmente una constante dividida por el tiempo de ejecución para un programa específico y entrada específica (como los MIPS relativos), estos son redundantes con el tiempo de ejecución, que es la principal medida de rendimiento. También, y para considerar algo distinto al tiempo de ejecución, suele ser tentador caracterizar una máquina con una única estimación en MIPS o MFLOPS sin ni siquiera nombrar programa alguno. Por último hay que decir que, como se ha indicado anteriormente, los MFLOPS no son una medida útil para todos los programas (hay algunos programas para los que no tiene sentido alguno hablar de MFLOPS).

EVALUACIÓN DEL RENDIMIENTO MEDIANTE PROGRAMAS

El candidato perfecto para evaluar un nuevo computador es aquel usuario que ejecuta día tras día los programas a utilizar en el mismo. Para evaluarlo, simplemente compararía el tiempo de ejecución de su *carga de trabajo* (*workload*), que tiene una mezcla de programas y órdenes del sistema operativo que el usuario corre en la máquina. Sin embargo, pocos usuarios pueden hacer esto. Normalmente hay que confiar en otros métodos para evaluar las máquinas y, con frecuencia, en otros evaluadores. Existen cuatro niveles de programas utilizados para la evaluación del rendimiento de un computador, que, según previsión, se enumeran en orden decreciente de precisión:

1. *Programas* (*Reales*) → Aún sin conocer qué fracción de tiempo se vaya a emplear en estos programas, sabe que se ejecutarán algunas veces para resolver problemas reales. Como ejemplos: los *compiladores de C*, el software de tratamiento de textos como *LaTeX* (lo forman un gran conjunto de macros de TeX (sistema tipográfico con órdenes muy elementales, de lenguaje de bajo nivel)), las herramientas de CAD como

Spice, etc. Los programas reales tienen entradas, salidas y opciones que un usuario puede seleccionar cuando está ejecutando el programa.

2. *Núcleos* (*Kernels*) → Se han hecho algunos intentos para extraer pequeñas piezas clave de programas reales y utilizarlas para evaluar el rendimiento. Como ejemplos más conocidos: *Livermore Loops* y *Linpack*. Al contrario que en los programs reales, ningún usuario corre los programas núcleo; únicamente se emplean para evaluar rendimiento. Los núcleos son adecuados para evaluar el rendimiento de una característica concreta de una máquina, aislada de las demás. Permite explicar las razones de las diferencias en rendimiento de distintas máquinas en programs reales.
3. *Benchmarks reducidos* (*toys*) → Normalmente tienen entre 10 y 100 líneas de código y producen un resultado conocido previamente por el usuario. Ejemplos de estos programas son: la *Criba de Eratóstenes*, *Puzzle* y *Clasificación Rápida* (Quicksort), con gran popularidad por ser pequeños, fáciles de introducir y ejecutar en cualquier computador.
4. *Benchmarks sintéticos* → Con una filosofía similar a los núcleos, los benchmarks sintéticos intentan determinar la frecuencia media de operaciones y operandos de un gran conjunto de programas. Ejemplos de estos programas, los más populares son: *Whetstone* y *Dhrystone*. Al igual que ocurre con los núcleos, ningún usuario ejecuta los benchmarks sintéticos porque no calculan nada que ningún usuario pueda utilizar. Los benchmarks sintéticos están más lejos de la realidad que los núcleos; mientras que los núcleos se extraen de programas reales, los benchmarks sintéticos se crean artificialmente para determinar un perfil medio de ejecución.

Cuando se va a utilizar un programa para evaluar el rendimiento es aconsejable comprobar que tenga entradas y no únicamente salidas. Si únicamente tiene salidas, siempre se va a obtener con él el mismo resultado. Aunque algunos programas reales utilizan pocas entradas, especialmente los programas de simulación y de aplicaciones de análisis numérico, cualquier programa real tiene alguna entrada.

Las ventas de los diferentes fabricantes de computadores dependen en gran medida del rendimiento de sus productos; hecho que provoca que se vuelquen en crear recursos para mejorar los resultados de la ejecución de los programas de evaluación del rendimiento. Estos recursos normalmente mejoran la ejecución de estos programas de evaluación pero no mejoran en absoluto la ejecución de los programas reales. Un caso extremo de esta situación ha consistido en emplear optimizaciones de compiladores que eran sensibles a los benchmarks; en lugar de realizar el análisis para que el compilador pudiera decidir adecuadamente si se podía aplicar la optimización, se utilizó un

prepocesador que comprobaba si se trataba en concreto de alguno de los benchmarks populares y, si coincidía con alguno de los que tenía en una lista predefinida realizaba las optimizaciones especiales; de esta forma obtenía unos buenos resultados de rendimiento. Los benchmarks van a ser tanto más vulnerables a estas prácticas cuanto más pequeños sean.

Pero, una vez conocidas los resultados erróneos a los que nos pueden condicir la utilización de núcleos, benchmarks reducidos y benchmarks sintéticos, ¿cuáles son las razones por las que no se ejecutan programas reales para medir el rendimiento?. Se pueden resumir en:

- Los núcleos y benchmarks son atractivos cuando se comienza un diseño ya que son lo suficientemente pequeños para simularlos fácilmente, incluso a mano. Hay que tener en cuenta que cuando se habla de una nueva máquina aún no se dispone de los compiladores para ella.
- Los benchmarks resultan atractivos por su fácil estandarización; los grandes programas resultan difíciles de estandarizar. Hay pocos resultados publicados para el rendimiento de grandes programas, en cambio hay muchos para los benchmarks.
- Los programas pequeños son más adecuados para el simulador de una arquitectura nueva puesto que tenían que correr normalmente sobre una máquina que era vieja y lenta
- Aunque las dos razones dadas anteriormente justifiquen en cierto modo el uso de núcleos y benchmarks, cuando se evalúan computadores ya existentes, ya no valen. Para estos computadores en funcionamiento, el uso de estos núcleos y benchmarks en la evaluación del rendimiento estaba justificado por:
  - en el pasado, los lenguajes de programación eran inconsistentes entre las distintas máquinas; y cada máquina tenía su propio sistema operativo. Esto hacía que fuese bastante complicado transportar de una máquina a otra un programa real.
  - normalmente no se disponía libremente del código fuente de los programas.

Aunque los núcleos, los benchmarks reducidos y los benchmarks sintéticos se utilizaron para intentar hacer comparaciones entre diferentes máquinas, actualmente no tiene sentido con la existencia de sistemas operativos estándar y la existencia de software distribuido libremente. Usar elementos inferiores a los programss reales

después de haber realizado los estudios iniciales de diseño, probablemente conduce a resultados erróneos y lleva por mal camino al diseñador.

COMPARACIÓN Y RESUMEN DE RENDIMIENTOS

La comparación de rendimientos de los computadores resulta una tarea difícil. Tomando como ejemplo la tabla 1.5, que muestra los tiempos de ejecución de dos programas y utilizando la definición dada para "*X es n% más rápida que Y*", se deducen las siguientes afirmaciones:

- El computador A es 900% más rápido que el computador B para el programa 1
- El computador B es 900% más rápido que el computador A para el programa 2
- El computador A es 1900% más rápido que el computador C para el programa 1
- El computador C es 4900% más rápido que el computador A para el programa 2
- El computador B es 100% más rápido que el computador C para el programa 1
- El computador C es 400% más rápido que el computador B para el programa 2

Programa	**Tiempo (s) Computador A**	**Teimpo (s) Computador B**	**Tiempo (s) Computador C**
1	1	10	20
2	1000	100	20
	t Total = 1001	t Total = 110	t Total = 40

Tabla 1.5. *Tiempos de ejecución de dos programas en tres máquinas.*

Si se toman individualmente cada una de estas afirmaciones puede ser útil pero, conjuntamente nada más que llevaría a confusión: el rendimiento relativo de los computadores A, B y C no está claro.

Considerando los tiempos totales de ejecución:

- El computador B es 810% más rápido que el computador A para los programas 1 y 2
- El computador C es 2402% más rápido que el computador A para los programas 1 y 2
- El computador C es 175% más rápido que el computador B para los programas 1 y 2

Si la carga de trabajo consistía en correr los programas 1 y 2 un número igual de veces, las afirmaciones anteriores predicen los tiempos de ejecución relativos para la carga de trabajo de cada máquina

Una media de los tiempos de ejecución, basada en el tiempo de ejecución total es la *media aritmética*:

$$tiempo\ ejecución_{media\ aritmética} = \frac{1}{n}\sum_{i=1}^{n} tiempo\ ejecución_i$$

$tiempo\ ejecución_i \rightarrow$ tiempo de ejecución del programa i-ésimo

$n \rightarrow$ número de programas de la carga de trabajo

Si el rendimiento se expresa como una frecuencia (por ejemplo MFLOPS), entonces la media del tiempo de ejecución total es la *media armónica*:

$$tiempo\ ejecución_{media\ armónica} = \frac{n}{\sum_{i=1}^{n} \frac{1}{velocidad_i}}$$

$velocidad_i \rightarrow$ Función de $\frac{1}{tiempo\ ejecución_i}$

$n \rightarrow$ número de programas de la carga de trabajo

Pero se plantea el problema de cuál es la mezcla adecuada de programas para la carga de trabajo. Como todos los programas no suponen el mismo peso para una carga de trabajo, se propone una *media aritmética ponderada*, asociándole un peso w_i a cada uno de los programas que forman parte de la carga de trabajo:

$$tiempo\ ejecución_{media\ aritmética\ ponderada} = \sum_{i=1}^{n} w_i \cdot tiempo\ ejecución_i$$

$w_i \rightarrow$ factor de peso asociado al programa i

$tiempo\ ejecución_i \rightarrow$ tiempo de ejecución del programa i-ésimo

$n \rightarrow$ número de programas de la carga de trabajo

Si el rendimiento se expresa como una frecuencia, entonces la media del tiempo de ejecución total es la *media armónica ponderada*:

$$tiempo\ ejecución_{media\ armónica\ ponderada} = \frac{1}{\sum_{i=1}^{n} \frac{w_i}{velocidad_i}}$$

$w_i \rightarrow$ factor de peso asociado al programa i

$velocidad_i \rightarrow$ Función de $\frac{1}{tiempo\ ejecución_i}$

$n \rightarrow$ número de programas de la carga de trabajo

Una segunda aproximación cuando hay una mezcla desigual de programas en la carga de trabajo consiste en *normalizar* los tiempos de ejecución con respecto a una máquina de referencia y después tomar la media de los tiempos de ejecución normalizados, análogamente a como se hizo en los MIPS relativos. En este caso, simplemente hay que multiplicar este número por su rendimiento en la máquina de referencia.

El *tiempo medio de ejecución normalizado* puede expresarse como una *media artimética* o como una *media geométrica*. Si se considera la *media geométrica*:

$$tiempo\ ejecución_{\substack{normalizado \\ media\ geométrica}} = \sqrt[n]{\prod_{i=1}^{n} Razón\ del\ tiempo\ de\ ejecución_i}$$

$Razón\ del\ tiempo\ de\ ejecución_i =$ tiempo de ejecución normalizado para el i-ésimo programa

$n \rightarrow$ número de programas de la carga de trabajo

Las medias geométricas tienen la siguiente propiedad:

$$\frac{Media\ geométrica(X_i)}{Media\ geométrica(Y_i)} = Media\ geométrica\left(\frac{X_i}{Y_i}\right)$$

En contraste con las medias aritméticas, las medias geométricas de los tiempos de ejecución normalizados son consistentes sin importar cuál sea la máquina de referencia. Por lo tanto, la media aritmética no se puede utilizar para promediar tiempos de ejecución normalizados.

El mayor inconveniente que tienen las medias geométricas de los tiempos de ejecución normalizados es que violan nuestro principio fundamental de medida de rendimiento: no predicen los tiempos de ejecución.

La solución ideal es medir una carga de trabajo real y ponderar los programas de acuerdo con su frecuencia de ejecución. Si esto no puede hecerse así, entonces se debe normalizar para que se emplee el mismo tiempo en cada programa sobre alguna máquina; al menos esto hace explícitos los pesos relativos y predice el tiempo de ejecución de una carga de trabajo con esa mezcla.

1.5. RESUMEN DE LA EVOLUCIÓN HISTÓRICA DEL CÁLCULO DEL RENDIMIENTO DE UN SISTEMA COMPUTADOR

Observando la historia, cada generación de computadores deja obsoletas las técnicas de evaluación del rendimiento de la generación anterior:

1. La medida original del rendimiento fue el *tiempo que se tardaba en realizar una operación individual*, como por ejemplo la suma, ya que la mayoría de las instrucciones tenían prácticamente el mismo tiempo de ejecución.
2. A partir de que los tiempos de ejecución de instrucciones no fueron uniformes, se calculó una *mezcla de instrucciones*, midiendo la frecuencia relativa de las instrucciones en un computador a través de muchos programas (por ejemplo, en 1970, la mezcla de Gibson). Se obtenía el *tiempo medio de ejecución por instrucción* (si se mide en ciclos de reloj, coincide con el *CPI medio*). A partir de aquí hubo un pequeño paso a los MIPS (son los inversos) que tienen como virtud el ser fáciles de comprender para el profano.
3. Cuando las CPUs se hicieron más sofisticadas (con jerarquías de memoria y segmentación) dejó de haber un único tiempo de ejecución por instrucción; los MIPS no se podían calcular a partir de la mezcla y del manual de características. Se empezaron a utilizar los *núcleos* (*kernels*) y los *benchmarks sintéticos*. Curnow y Wichmann (1976) crearon el benchmark sintético Whetstone para medir programas científicos escritos en Algol 60; este programa se redactó en FORTRAN. Buscando objetivos similares McMahon y los investigadores del Lawrence Livermore Laboratory (1986) realizaron el Livermore FORTRAN Kernels para la evaluación de supercomputadores. Estos kernels estaban construídos con bucles de programas reales. La noción de MIPS relativos (tomando como referencia el VAX-11/780) se presentó como una forma de resucitar la estimación fácilmente comprensible de los MIPS. Los eruditos en materia de rendimeinto de sistemas computadores han redefinido los MIPS como "*indicación sin sentido de la velocidad del procesador*" (*Meaningless Indication of Processor Speed*) o también como "adoctrinamiento sin sentido para vendedores agresivos" (*Meaningless Indoctrination by Pushy*

Salerpersons). Actualmente, el significado de los MIPS en términos de marketing no es el de los MIPS nativos sino "*número de veces más rápido que el VAX-11/780*".

4. Los años 70 y 80 marcaron el crecimiento de la industria de los supercomputadores, que fueron definidos de esa forma por el alto rendimiento en programas intensivos en punto flotante. El tiempo medio de instrucción y los MIPS no eran métricas apropiadas, de ahí la invención de los MFLOPS. Desgraciadamente, los grupos de marketing de las empresas empezaron a citar los MFLOPS máximos sin tener en cuenta los programas usados para calcularlos.
5. Se proponen diversas medidas para promediar el rendimiento: McMahon (1986) recomienda la media armónica para promediar los MFLOPS. Flemming y Walace (1986) defienden los méritos de la media geométrica. En 1988 Smith presenta una réplica a las anteriores.
6. Entre la comunidad de la computación se extiende la distinción entre arquitectura e implementación y se busca la forma de evaluar independientemente una y otra. Se hace un estudio en la Universidad de Carnegie-Mellon, que propone tres medidas cuantitativas:
   (a) S → Número de bytes del código del programa
   (b) M → Número de bytes transferidos entre la memoria y la CPU, tanto para instrucciones como para datos, durante la ejecución de un programa

   (S mide el tamaño del código en tiempo de compilación, mientras que M mide el tráfico de memoria durante la ejecución de un programa)

   (c) R → Número de bytes transferidos entre registros en un modelo canónico de CPU
7. En 1988 se forma el grupo Cooperativa de Evaluación del Rendimiento de Sistemas (System Performance Evaluation Cooperative) o SPEC, para la evaluación del rendimiento de los sistemas computadores. Este grupo tiene representantes de muchas compañias de computadores; los fundadores fueron. Apollo/hewlett-Packard, DEC, MIPS y Sun. Todos ellos llegaron al acuerdo de ejecutar en sus máquinas un conjunto de programas y entradas reales para evaluar el rendimiento. Es posible esta forma de evaluación gracias a los sistemas operativos portables (UNIX) y los lenguajes de alto nivel; ahora los compiladores forman parte de los programas para la evaluación del rendimiento. La versión 1.0 de SPEC contenía programas con código entero (Gcc, Expreso, Li, etc.) y en coma flotante (Spice, Doduc, etc.).
8. Un esfuerzo similar a SPEC, llamado Perfect Club, vincula a las universidades y compañías interesadas en el cálculo paralelo. En lugar de forzar a correr el código de

programas secuenciales existentes, Perfect Club incluye programas y algoritmos, y permite a los miembros escribir nuevos programas en nuevos lenguajes, que pueden ser necesarios para las nuevas arquitecturas. Los miembros de Perfect Club también pueden sugerir nuevos algoritmos para resolver problemas importantes.