Departamento de Arquitectura Instituto de Computación

Universidad de la República

Montevideo - Uruguay

Notas de Teórico

Evaluación de Rendimiento

Arquitectura de Computadoras

(Versión 1.1 - 2014)

EVALUACIÓN DE RENDIMIENTO

1 Introducción

Al evaluar el hardware de un sistema o al determinar los requerimientos para un nuevo sistema, el rendimiento es uno de los parámetros clave a considerar junto con el costo, el tamaño, la seguridad, la fiabilidad e incluso en algunos casos el consumo de energía.

Es complejo realizar comparaciones significativas de rendimiento entre diferentes procesadores, o incluso entre procesadores de una misma familia. La velocidad es bastante menos importante que como un procesador se comporta al ejecutar una aplicación dada. Desafortunadamente, el rendimiento de una aplicación depende no solo de la velocidad del procesador sino también de el set de instrucciones, el lenguaje en el que se implementa, la eficiencia del compilador, y la habilidad con la que se programó la aplicación.

En este capítulo estudiaremos algunas métricas tradicionales para la velocidad del procesador y se examinarán los enfoques mas comunes para evaluar el rendimiento de un sistema. El objetivo es lograr ver a través de la "niebla" del marketing y poder tomar decisiones inteligentes al momento de elegir un sistema computacional.

2 El reloj del sistema

Las operaciones realizadas por un procesador, tales como ir a buscar una instrucción a memoria (fecth), decodificar la instrucción (decode), realizar una operación aritmética (execute), están gobernadas por un reloj del sistema. Típicamente, todas las operaciones comienzan con el pulso del reloj. Entonces, al nivel mas básico, la velocidad del procesador esta dada por la frecuencia del pulso producida por el reloj, medido en ciclos por segundo o Hertz (Hz). Por ejemplo, un procesador de 1 Ghz recibe mil millones de pulsos por segundo. La tasa de pulsos se denomina en general **frecuencia o velocidad del reloj**, un pulso es comúnmente referido como un **ciclo de reloj** y el tiempo entre pulsos se conoce como **tiempo de ciclo o período.** Es importante hacer notar que la frecuencia del reloj no es elegida arbitrariamente, sino que debe ser la apropiada para el diseño físico del procesador.

La ejecución de una instrucción requiere de un conjunto de pasos discretos tal como se menciono previamente. Por lo tanto, la mayoría de las instrucciones en la mayoría de los procesadores requieren de múltiples ciclos de reloj para ser completadas. Algunas instrucciones pueden tomar solo algunos pocos ciclos, mientras que otras requieren decenas de ciclos. Esto implica que una simple comparación entre frecuencias de reloj en diferentes procesadores no nos cuenta la historia completa sobre el rendimiento.

3 Tiempo

El tiempo es la medida del rendimiento del computador: el computador que realiza la misma cantidad de trabajo en el mínimo tiempo es el más rápido. El **tiempo de ejecución** de un programa se mide en segundos por programa. El rendimiento se mide frecuentemente como una frecuencia de eventos por segundo, ya que tiempo más bajo significa mayor rendimiento. Tendemos a desdibujar esta distinción y hablamos del rendimiento como tiempo o como velocidad, incluyendo refinamientos como mejora del rendimiento en lugar de utilizar los adjetivos mayor (para velocidades) o menor (para el tiempo). Es decir:

$$\text{Rendimiento}(x) = \frac{1}{\text{Tiempo de Ejecución}(x)}$$

Pero el tiempo se puede definir de formas distintas dependiendo de lo que queramos contar. La definición más directa de tiempo se denomina tiempo de reloj, tiempo de respuesta, o **tiempo transcurrido (elapsed time)**. Esta es la latencia para completar una tarea, incluyendo accesos a disco, accesos a memoria, actividades de entrada/salida, gastos del sistema operativo, es decir, todo. Sin embargo, como en multiprogramación la CPU trabaja sobre otro programa mientras espera las E/S y, no necesariamente puede minimizar el tiempo transcurrido de un programa; es necesario un término que tenga en cuenta esta actividad. El **tiempo de CPU** reconoce esta distinción y mide el tiempo que la CPU está calculando, sin incluir el tiempo de espera para las E/S o para ejecutar otros programas. (Obviamente, el tiempo de respuesta visto por el usuario es el tiempo transcurrido del programa, no el tiempo de CPU).

En la discusión actual, se mantiene la distinción entre el rendimiento basado en el tiempo transcurrido y el basado en el tiempo de CPU. El término **rendimiento del sistema** se utiliza para referenciar el tiempo transcurrido**,** mientras que el **rendimiento de la CPU** se refiere al tiempo de CPU**.** En el curso nos concentraremos en el rendimiento de la CPU.

4 Rendimiento de la CPU

Como vimos anteriormente, la mayoría de los computadores se construyen utilizando un reloj que funciona a una frecuencia constante. El tiempo de CPU para un programa puede expresarse entonces de dos formas:

$$\text{Tiempo de CPU} = \text{Ciclos de reloj de CPU para un programa} * \text{Duración del ciclo de reloj}$$

o

$$\text{Tiempo de CPU} = \frac{\text{Ciclos de reloj de CPU para un programa}}{\text{Frecuencia de reloj}}$$

Obsérvese que no tendría sentido mostrar el tiempo transcurrido como 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.

Además del número de ciclos de reloj para ejecutar un programa, también podemos contar el número de instrucciones ejecutadas (el **recuento de** instrucciones). Si conocemos el número de ciclos de reloj y el recuento de instrucciones podemos calcular el número medio de **ciclos de reloj por instrucción (CPI)**:

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

Esta medida del rendimiento de la CPU proporciona una nueva percepción de los diferentes estilos de repertorios de instrucciones e implementaciones. Al transponer el «recuento de instrucciones» en la fórmula anterior, los ciclos de reloj pueden definirse como recuento de instrucciones * CPI. Esto nos permite utilizar el CPI en la fórmula del tiempo de ejecución:

$$\text{Tiempo de CPU} = \text{Recuento de instrucciones} * CPI * \text{Duración del ciclo de reloj}$$

o

$$\text{Tiempo de CPU} = \frac{\text{Recuento de instrucciones} * CPI}{\text{Frecuencia de reloj}}$$

Expandiendo la primera fórmula en las unidades de medida mostradas obtenemos:

$$\text{Timepo de CPU} = \frac{\text{Segundos}}{\text{Programa}} = \frac{\text{Instrucciones}}{\text{Programa}} * \frac{\text{Ciclos de Reloj}}{\text{Instrucción}} * \frac{\text{Segundos}}{\text{Ciclo de Reloj}}$$

Como demuestra esta fórmula, el rendimiento de la CPU depende de tres características: ciclo de reloj (o frecuencia), ciclos de reloj por instrucción, y recuento de instrucciones. No se puede cambiar ninguna de ellas sin tener en cuenta las demás, ya que las tecnologías básicas involucradas al cambiar una característica también son interdependientes:

- Frecuencia de reloj - Tecnología de hardware y organización
- CPI - Organización y arquitectura a nivel lenguaje máquina
- Recuento de instrucciones - Arquitectura del nivel lenguaje máquina y tecnología de compiladores

Uno de los errores mas habituales es considerar que uno o dos de estos factores son determinantes del rendimiento, por ejemplo, un caso típico es solo tener en cuenta la frecuencia de reloj. La siguiente tabla muestra la relación entre estos valores.

	Recuento de Inst.	CPI	Ciclo de Reloj
Programa	X		
Compilador	X	(X)	
Set de inst.	X	X	
Organización		X	X
Tecnología			X

A veces es útil, al diseñar la CPU, calcular el número total de ciclos de reloj de la CPU como:

$$\text{Ciclos de reloj de la CPU} = \sum_{i=1}^{n} (CPI_i * I_i)$$

donde I_i , representa el número de veces que se ejecuta la instrucción i en un programa con un set e instrucciones con n instrucciones y CPI_i representa el número medio de ciclos de reloj para la instrucción i. Esta forma puede utilizarse para expresar el tiempo de CPU como:

$$\text{Tiempo de CPU} = \sum_{i=1}^{n} (CPI_i * I_i) * \text{Duración del ciclo de reloj}$$

y el CPI global como:

$$CPI = \frac{\sum_{i=1}^{n} (CPI_i * I_i)}{\text{Recuento de instrucciones}} = \sum_{i=1}^{n} (CPI_i * \frac{I_i}{\text{Recuento de instrucciones}})$$

La última forma del cálculo de CPI multiplica cada CPI individual, por la fracción de ocurrencias de esa instrucción en un programa.

CPI_i , debe medirse, y no calcularse a partir de una tabla al final del manual de referencia, ya que debe incluir fallos de cache y demás ineficiencias del sistema de memoria.

Tener siempre en cuenta que la medida real del rendimiento del computador es el tiempo. Cambiar el repertorio de instrucciones para disminuir el recuento de instrucciones; por ejemplo, puede conducir a una organización con un ciclo de reloj de mayor duración que contrarresta las mejoras en el recuento de instrucciones. Cuando se comparan dos máquinas se deben examinar los tres componentes para comprender el rendimiento relativo.

5 MIPS

En la búsqueda de una medida estándar del rendimiento de los computadores se ha adoptado una serie de medidas populares, con el resultado de que algunos términos inocentes se han secuestrado de un entorno bien definido y forzado a un servicio para el cual nunca fueron pensados. La posición que tomamos en el curso es que la única medida fiable y consistente del rendimiento es el tiempo de ejecución de los programas reales, y que las demás alternativas propuestas al tiempo como métrica o a los programas reales como «items» medidos han conducido, eventualmente, a afirmaciones erróneas o incluso a errores en el diseño de los computadores.

Una alternativa al tiempo como métrica son los **MIPS**, o **millones de instrucciones por segundo**. Para un programa dado, los MIPS son sencillamente:

$$MIPS = \frac{\text{Recuento de instrucciones}}{\text{Tiempo de ejecución} * 10^6} = \frac{\text{Frecuencia de reloj}}{CPI * 10^6}$$

Algunos encuentran adecuada la fórmula de más a la derecha, ya que la frecuencia de reloj es fija para una máquina y el CPI, habitualmente, es un número pequeño, de forma distinta a la cuenta de instrucciones o al tiempo de ejecución. La relación de los MIPS con el tiempo es:

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

Como los MIPS son una frecuencia de operaciones por unidad de tiempo, el rendimiento puede especificarse como el inverso del tiempo de ejecución, de forma que máquinas más rápidas tendrán una mayor frecuencia de MIPS. La buena noticia sobre los MIPS es que son fáciles de comprender, especialmente por un cliente, y máquinas más rápidas significan un mayor número de MIPS, lo cual coincide con la intuición. El problema, cuando se utilizan los MIPS como medida para hacer comparaciones, es triple:

- Los MIPS son dependientes del repertorio de instrucciones, lo cual hace difícil la comparación de los MIPS de computadores con diferentes repertorios de instrucciones;
- Los MIPS varían entre programas en el mismo computador; y lo más importante,
- ¡LOS MIPS pueden variar inversamente al rendimiento!

El ejemplo clásico, del último caso, es la variación de los MIPS en una máquina con hardware opcional de punto flotante. Como, generalmente, se emplean más ciclos de reloj por instrucción en punto flotante que por instrucción entera, los programas en punto flotante que utilizan el hardware opcional en lugar de las rutinas software de punto flotante emplean menos tiempo, pero tienen una **menor** frecuencia de MIPS. El software de punto flotante ejecuta instrucciones más simples, dando como resultado una mayor frecuencia de MIPS, pero se ejecuta tantas veces que el tiempo global de ejecución es mayor.

6 Elección de programas para evaluar el rendimiento

Un usuario de computadores que ejecuta los mismos programas día tras día debería ser el candidato perfecto para evaluar un nuevo computador. Para evaluar un nuevo sistema simplemente compararía el tiempo de ejecución de su **carga de trabajo (workload)** -la mezcla de programas y órdenes del sistema operativo que los usuarios corren en una máquina. Sin embargo, pocos están en esta feliz situación. La mayoría debe confiar en otros métodos, para evaluar las máquinas, y, con frecuencia, en otros evaluadores, esperando que estos métodos predigan el rendimiento de la nueva máquina. Hay cuatro niveles de programas utilizados en estas circunstancias, listados a continuación en orden decreciente de precisión de la predicción.

1. **Programas (Reales)**. Aunque el comprador puede no conocer qué fracción de tiempo se emplea en estos programas, sabe que algunos usuarios los ejecutarán para resolver problemas reales. Ejemplos son compiladores de *C,*software de tratamiento de textos como TeX o Word, y herramientas *CAD* como AutoCAD. 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. Livermore Loops y Linpack son los ejemplos mejor conocidos. De forma distinta a los programas reales, ningún usuario puede correr los programas «núcleo»; únicamente se emplean para evaluar el rendimiento. Los «núcleos» son adecuados para aislar el rendimiento de las características individuales de una máquina para explicar las razones de las diferencias en los rendimientos de programas reales.
3. **Benchmarks reducidos (toys)**. Los benchmarks reducidos, normalmente, tienen entre 10 y 100 líneas de código y producen un resultado que el usuario conoce antes de ejecutarlos. Programas como la Criba de Eratóstenes, Puzzle, y Clasificación Rápida *(Quicksort)* son populares porque son pequeños, fáciles de introducir y de ejecutar casi en cualquier computador. El mejor uso de estos programas es empezar asignaciones de programación.
4. **Benchmarks Sintéticos**. Análogos en filosofía a los «núcleos», los benchmarks sintéticos intentan determinar la frecuencia media de operaciones y operandos de un gran conjunto de programas. Whetstone y Dhrystone son benchmarks sintéticos populares. 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, en efecto, aún más lejos de la realidad porque el código de los «núcleos» se extrae de programas reales, mientras que el código sintético se crea artificialmente para determinar un perfil medio de ejecución. Los benchmarks sintéticos no son partes de programas reales, mientras que los demás pueden serlo.

La desventaja de programas estandarizados para medir rendimiento es que pueden ser “violados”. Debido a que las compañías de computadores florecen o quiebran dependiendo del precio/rendimiento de sus productos con respecto a los demás productos del mercado, se dispone de enormes recursos para mejorar el rendimiento de los programas más utilizados en evaluar rendimientos. Tales presiones pueden desviar los esfuerzos de las ingenierías del hardware y software para añadir optimizaciones que mejoren el rendimiento de los programas sintéticos reducidos, o de los núcleos, pero no de los programas reales. Un ejemplo extremo de esta ingeniería empleó 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, una persona de una compañía recién creada utilizó un preprocesador que exploraba las palabras claves del texto para tratar de identificar los benchmarks examinando el nombre del autor y el nombre de una subrutina esencial. Si la exploración confirmaba que este programa estaba en una lista predefinida, se realizaban las optimizaciones especiales. Esta máquina consiguió un cambio brusco en el rendimiento -al menos de acuerdo con aquellos benchmarks. No obstante estas optimizaciones no eran aplicables a programas que no estuviesen en la lista, sino que eran inútiles para un código idéntico con algunos cambios de nombre.

¿Por qué no se ejecutan programas reales para medir el rendimiento? Los núcleos y benchmarks reducidos son atractivos cuando se comienza un diseño ya que son lo suficientemente pequeños para simularlos fácilmente, incluso a mano. Son especialmente tentadores cuando se inventa una nueva máquina, porque los compiladores no están disponibles hasta mucho más tarde. Los pequeños benchmarks también se estandarizan más fácilmente, mientras que los grandes programas son más difíciles de estandarizar; de

aquí que haya numerosos resultados publicados para el rendimiento de pequeños benchmarks pero pocos para los grandes. Además, tan importante como el resto es que a la hora informar sobre las medidas de rendimiento la reproducibilidad es esencial y un benchmark lo permite fácilmente.

Este hecho causó que algunos proveedores de benchmarks especifiquen reglas bajo las cuales los compiladores deben operar, como veremos a continuación.

6.1 Benchmark suites

Es bastante popular actualmente juntar colecciones de benchmarks para lograr medir el rendimiento de procesadores con una variedad de aplicaciones. Por supuesto, tales suites son tan buenas como lo sean cada benchmark individual. Sin embargo, una ventaja importante es que la debilidad de un benchmark is contrarrestada por la presencia de otro benchmark.

Uno de los intentos mas exitosos para crear estandarizaciones de benchmark suites es el SPEC (Standard Perfromance Evaluation Corporation). Así como la industria de la computación a evolucionado con el tiempo, también lo ha hecho la necesidad de diferentes suites de benchmarks, y por lo tanto existen SPEC benchmarks para cubrir diferente clase de aplicaciones. Todos los suites de SPEC así como sus resultados se pueden encontrar en www.spec.org.

7 Mejorando el rendimiento

Al considerar el rendimiento de un sistema, los diseñadores buscan maneras de mejorar el rendimiento realizando mejoras en la tecnología o cambiando el diseño. Ejemplos incluyen el uso de procesadores paralelos, el uso de jerarquía de memoria, y mejoras en el tiempo de acceso a memoria o entrada/salida con nuevas tecnologías. En todos estos casos es importante notar que una mejora en un aspecto de la tecnología o diseño no resulta en la misma mejora en el rendimiento total.

Quizá el principio más importante y generalizado del diseño de computadores sea acelerar el caso común: al realizar un diseño, favorecer el caso frecuente sobre el infrecuente. Este principio también se aplica cuando se determina cómo emplear recursos, ya que el impacto de hacer alguna ocurrencia más rápida es mucho mayor si la ocurrencia es frecuente. Mejorar el evento frecuente en lugar del evento raro, evidentemente, también ayudará a aumentar el rendimiento. Además, el caso frecuente es, a menudo, más simple y puede realizarse de forma más rápida que el caso infrecuente. Al aplicar este sencillo principio, hemos de decidir cuál es el caso frecuente y cómo se puede mejorar el rendimiento haciendo este caso más rápido. Este principio es expresado sucintamente por la ley de Amdahl.

7.1 Ley de Amdahl

La ley de Amdahl fue propuesta por Gene Amdahl y trata sobre la aceleración potencial de un programa usando múltiples procesadores comparado con el uso de un solo procesador.

En nuestro caso nos interesa la generalización de la ley de Amdahl para evaluar cualquier mejora en diseño o tecnología de un sistema computacional. Esta generalización 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.

Pero, ¿qué es la aceleración?. Supongamos que podemos hacer una mejora en una máquina que cuando se utilice aumente su rendimiento. La aceleración *(speedup)* es la relación:

$$\text{Aceleración} = \frac{\text{Rendimiento de la tarea completa utilizando la mejora cuando sea posible}}{\text{Rendimiento de la tarea completa sin utilizar la mejora}}$$

ó

$$\text{Aceleración} = \frac{\text{Tiempo de ejecución de la tarea sin utilizar la mejora}}{\text{Tiempo de ejecución de la tarea completa utilizando la mejora cuando sea posible}}$$

La aceleración nos indica la rapidez con que se realizará una tarea utilizando una máquina con la mejora con respecto a la máquina original.

La generalización de la Ley de Amdahl nos da una forma rápida de calcular la aceleración, que depende de dos factores:

1. La fracción del tiempo de cálculo de la máquina original que pueda utilizarse para aprovechar la mejora. Este valor, que llamaremos $\text{Fracción}_{\text{afectada}}$ es siempre menor o igual que 1.

2. 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. Este valor es el tiempo del modo original con respecto al tiempo del modo mejorado y es siempre mayor que 1. Llamaremos a este valor $\text{Aceleración}_{\text{afectada}}$

El tiempo de ejecución utilizando la máquina original con el modo mejorado será el tiempo empleado utilizando la parte no mejorada de la máquina más el tiempo empleado utilizando la parte mejorada. Por lo tanto:

$$T_{\text{old}} = T_{\text{no afectado}} + T_{\text{afectado old}}$$

$$T_{\text{new}} = T_{\text{no afectado}} + T_{\text{afectado new}}$$

$$T_{\text{new}} = T_{\text{no afectado}} + \frac{T_{\text{afectado old}}}{\text{aceleración}_{\text{afectada}}} \quad \text{donde aceleración}_{\text{afectada}} = \frac{T_{\text{afectado old}}}{T_{\text{afectado new}}}$$

$$T_{\text{new}} = (1-F) * T_{\text{old}} + \frac{F * T_{\text{old}}}{\text{aceleración}_{\text{afectada}}}$$

$$\text{aceleración}_{\text{total}} = \frac{T_{\text{old}}}{T_{\text{new}}} = \frac{1}{(1-F) + \frac{F}{\text{aceleración}_{\text{afectada}}}}$$

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 del cálculo disminuye tal como se van añadiendo mejoras. Un corolario importante de 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 recíproco de 1 menos esa fracción. Es decir:

$$\lim_{\text{aceleración}_{\text{afectada}} \to \infty} \text{aceleración}_{\text{total}} = \frac{1}{(1-F)}$$

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, midiésemos el tiempo después 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.