Universidad Europea

LAUREATE INTERNATIONAL UNIVERSITIES

Medidas de rendimiento

Contenido

Presentación

Un buen ingeniero o arquitecto de computadoras debe conocer las **partes básicas de una computadora digital**, así como las distintas **arquitecturas avanzadas** que normalmente se utilizan para procesamiento intenso.

Estos conocimientos permiten al ingeniero, desde un punto de vista práctico, conocer a fondo las características y el **funcionamiento del hardware**. La capacidad de interpretar la estructura, las características y las capacidades de una arquitectura, permitirán medir el rendimiento de un sistema y, en la medida de lo posible, mejorarlo. También es importante saber comparar dos arquitecturas distintas y así poder escoger el sistema más apropiado para cada ámbito de aplicación.

Objetivos

Los objetivos que se pretenden alcanzar al terminar este recurso son los siguientes:

- Avanzar en los conocimientos sobre los **componentes hardware** de una computadora y conocer **nuevos conceptos** sobre medidas de rendimiento.
- Ser capaz de **medir y calcular** el rendimiento de los computadores y de componentes específicos.
- Ser capaz de comparar el rendimiento de ordenadores de **diferentes familias** entre sí. Igualmente, ser capaz de comparar el rendimiento de distintas configuraciones de un mismo computador.
- Reconocer la **evolución** y mejora de los computadores.
- Conocer qué es un ***benchmark*** o **programas de pruebas**, y conocer los estándares actuales y cómo utilizarlos.

Niveles de descripción

Un computador por sí mismo es un sistema complejo que engloba varios niveles de estudio a nivel software y hardware. En la actualidad, la mayoría de los sistemas informáticos implican **arquitecturas complejas de computadores** que combinan los componentes básicos definidos en el **modelo Von Neumann**.

Un ingeniero informático debe ser capaz de definir, diseñar, desarrollar y mantener sistemas y arquitecturas informáticas complejas. Para diseñar un sistema informático será necesario, por tanto, integrar la parte hardware con la parte software, así como establecer los mecanismos más adecuados de c**omunicación** entre los distintos componentes y sistemas.

En la siguiente tabla se detallan los niveles de abstracción de una computadora.

Niveles de abstracción de una computadora

Los niveles de abstracción definen los **distintos niveles de estudio** que un ingeniero informático debe conocer para poder entender el funcionamiento de una computadora en su conjunto. Serían los siguientes:

Capas	Niveles de abstracción	Componentes y herramientas
Software de aplicación	Generadores de programas	• Entornos IDE. • Lenguajes de programación.
	Programas de aplicación	• Ofimática (Ms Office (Microsoft, 2016), Open Office (Apache, 2016), multimedia, diseño, comunicaciones, etc.
Software de sistema	Compiladores e intérpretes.	• Compilación, enlazado, ensamblado, traducción
	Sistemas operativos.	• Sistemas operativos: UNIX (The Open Group, 2016), Windows (Microsoft, 2016), OS X (Apple, 2016), software de gestión de ficheros, memoria, procesos, etc.
Interfaz software/hardware	Arquitectura del conjunto de instrucciones	• Código máquina, arquitectura RISC y CISC.
Hardware	Organización de componentes hardware.	• Estructura básica: modelo Von Neumann.

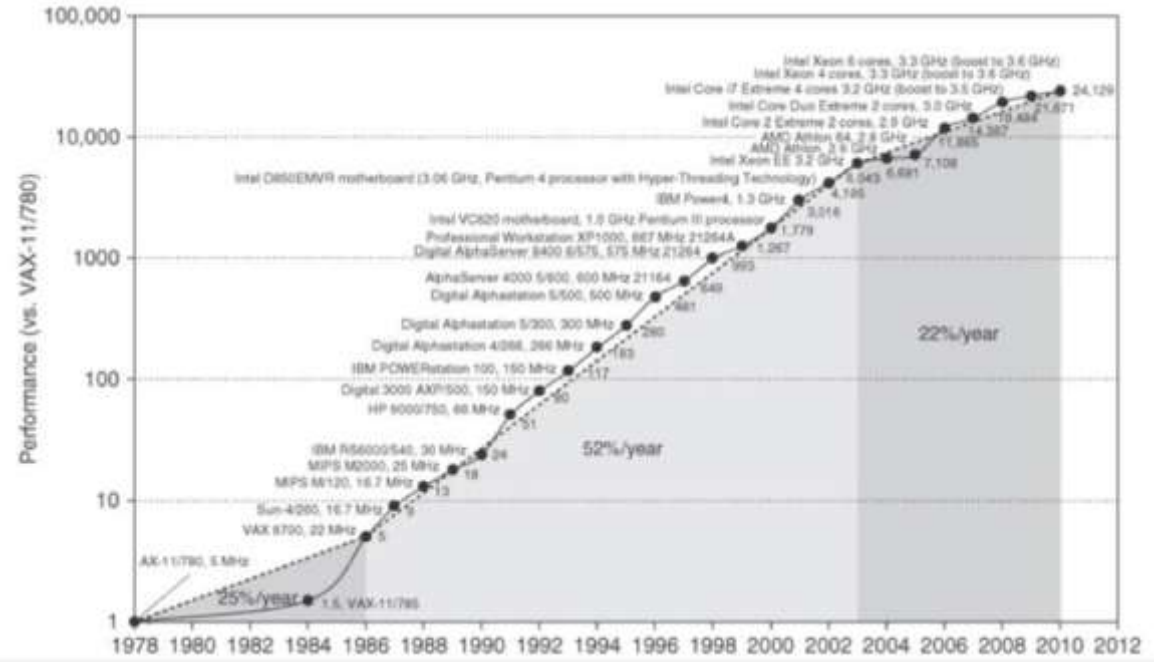
		• Arquitecturas avanzadas. Transferencias.
	Nivel digital.	• Circuitos secuenciales. Circuitos combinacionales
	Nivel físico.	• Electrónico: inversor, multivibrador. • Componentes: transistor, metal, diodo.

El diseño de arquitecturas avanzadas se centra en estos **dos niveles**:

Arquitectura del conjunto de instrucciones	Este nivel define el puente entre el hardware y el software y se centra en el **comportamiento de la máquina**. Es decir, qué hace la máquina. En este nivel, además, se define cuál será el juego de instrucciones de la máquina a utilizar. Esta decisión determinará el **diseño hardware del computador**. El tamaño de las **instrucciones y los datos** determinarán el tamaño de los registros de la máquina, el tamaño de las celdas de almacenamiento de memoria y el número de hilos de los componentes de interconexión (por ejemplo, el bus de instrucciones y el bus de datos).
Organización de componentes hardware	La arquitectura de computadores determina la organización de los componentes hardware y debe conocer el funcionamiento interno de cada componente. El objetivo principal del arquitecto de computadores es construir una máquina lo más **eficiente** posible al **menor coste** posible, teniendo en cuenta el **ámbito de aplicación** de la máquina (genérico o específico) y las limitaciones del **presupuesto** asignado.

Evolución de los computadores

En la siguiente gráfica podemos observar cómo ha evolucionado el **rendimiento** de los computadores en los últimos años:



Evolución del rendimiento de los computadores. Fuente: Patterson y Heneessy, 2011.

En este gráfico se muestra la evolución del rendimiento de distintos procesadores, en comparación con la máquina VAX 11/780, que fue utilizada como **máquina de referencia** por los *benchmarks* SPEC hasta el año 2000.

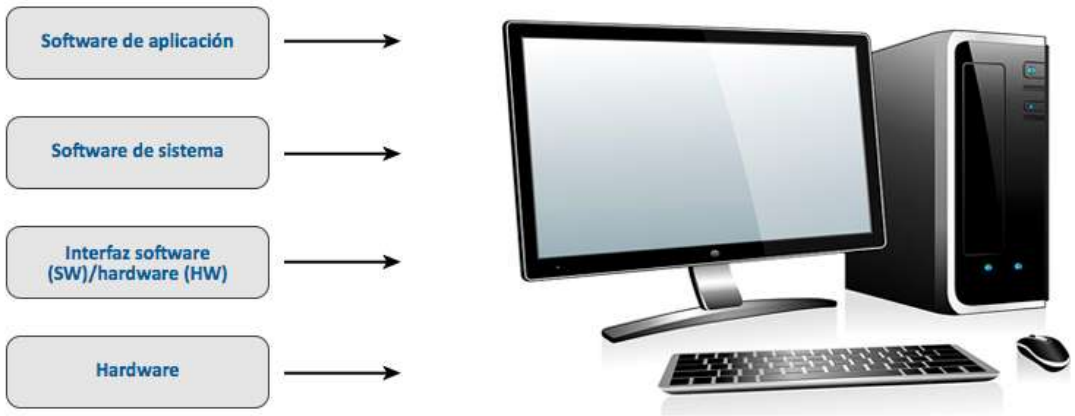
A continuación, se analiza en detalle el gráfico con la evolución del rendimiento de los computadores:

De 1986 a 2003
Entre el año 1986 y el 2003 se produjo la gran explosión de los ordenadores personales: • La tecnología evolucionó de tal forma que el rendimiento de los ordenadores mejoraba a un ritmo de 52% cada año. • El precio de los ordenadores bajó espectacularmente. • Los ordenadores personales llegaron a los hogares y las ventas de ordenadores se dispararon.
De 2003 a 2011
Sin embargo, en la última década, la evolución ha sido en torno al 22% anual, debido a las **limitaciones de consumo de potencia** y a que la introducción de los multiprocesadores no mejora el rendimiento proporcionalmente al número de procesadores.

Medidas de rendimiento versus niveles de descripción

Dependiendo del nivel de abstracción o de descripción en el que queramos **mejorar la máquina**, deberemos fijarnos en una u otra medida de rendimiento.

A continuación, se muestran algunas **medidas de rendimiento**, clasificadas según el nivel de abstracción y la característica que queramos mejorar:



Software de aplicación

Niveles de abstracción	Medidas de rendimiento
Generadores de programas.	• Número de Instrucciones de alto nivel. • Complejidad del algoritmo (Ω).
Programas de aplicación	• Tiempo de ejecución (Tej). • Tiempo de respuesta de usuario. • Tiempo de CPU (T_{CPU}). • Operaciones por segundo. • Rendimiento (*performance*). • Productividad o *throughput*. • Tiempo de respuesta.

Software de sistema

Niveles de abstracción	Medidas de rendimiento
Compiladores e intérpretes.	• Número de instrucciones máquina por programa (N).
Sistemas operativos	• Número de programas en paralelo. • Número de *threads* en paralelo.

Interfaz software (SW)/hardware (HW)

Niveles de abstracción	Medidas de rendimiento
Arquitectura del conjunto de instrucciones.	• Ciclos por instrucción (CPI). • Instrucciones por segundo (MIPS). • Instrucciones de coma flotante por segundo (MFLOPS).
Organización de componentes hardware	• Tiempo de acceso a memoria (Tacc, TAM). • Latencia. • Ancho de banda (megabytes/segundo). • Transacciones Por Segundo (TPS).

Hardware

Niveles de abstracción	Medidas de rendimiento
Nivel digital.	• Frecuencia de reloj (F). • Periodo o ciclo de reloj (t, T). • Inferencias lógicas por segundo (KLIPS).
Nivel físico.	• Número de transistores por componente.

Tiempo de ejecución

Durante todo este recurso, hablaremos de muchas medidas de rendimiento como las que enumerábamos en el apartado anterior.

Estudiaremos algunas de ellas, aprendiendo su definición y memorizando fórmulas para calcularlas. Sin embargo, cuando llegue la hora de medir el **rendimiento global** de un computador, la prueba definitiva sobre si una arquitectura es óptima o si hemos mejorado sus prestaciones lo suficiente, solo nos fijaremos en una medida: el **tiempo de ejecución**.

El tiempo de ejecución de programas reales es la única medida de rendimiento fiable.

Tiempo de ejecución
Definimos el tiempo de ejecución como el **tiempo de respuesta del sistema** desde que se ejecuta un programa hasta que se obtiene una respuesta.

El tiempo de ejecución es, por tanto, la **medida de la verdad**. De su resultado dependerá que todos los recursos utilizados para la compra y construcción de componentes para diseñar la arquitectura, diseñar algoritmos óptimos o buscar mejoras en las prestaciones, hayan merecido o no la pena.

Hay que distinguir entre el tiempo de ejecución de usuario (Tej), y el tiempo de ejecución del procesador (T_{CPU}):

 Tiempo de ejecución de usuario (Tej)	Tiempo que el usuario tarda en recibir una **respuesta del sistema** desde que ejecuta un programa hasta que recibe el resultado por la pantalla o por la interfaz de usuario correspondiente.
 Tiempo de ejecución de procesador (T_{CPU})	Tiempo que medimos por software desde que se **inicia la ejecución** de un programa hasta que se **genera el resultado** buscado como salida.

Rendimiento

Como ya anunciara Einstein en su **Teoría de la Relatividad**: "dos observadores que se mueven relativamente uno al lado del otro con distinta velocidad, a menudo obtendrán diferentes medidas del tiempo".

Cuando tratamos de evaluar las prestaciones de una arquitectura de computadores ocurre algo similar. Cuando decimos que el rendimiento de un computador es óptimo, ¿respecto a qué medida? ¿Comparado con qué?

El rendimiento de un computador es **relativo**, dependiendo del fin para que sea utilizado y de cómo de bueno sea comparado con máquinas similares que fueron construidas para el mismo fin o similar.

Para medir el rendimiento de un computador, ya tenemos claro que la única medida fiable es el tiempo de ejecución, pero no hemos delimitado cuál es el programa que ejecutamos, ni cuál es la medida de tiempo que se espera. Para demostrar que una computadora es óptima, debemos ejecutar un **programa real** (o un programa de prueba de complejidad similar), y comparar los tiempos de ejecución con una computadora similar. Cuanto menor sea el tiempo de ejecución, mejor es nuestro sistema.

Reducir el tiempo de ejecución, mejora el r**endimiento** de un sistema.

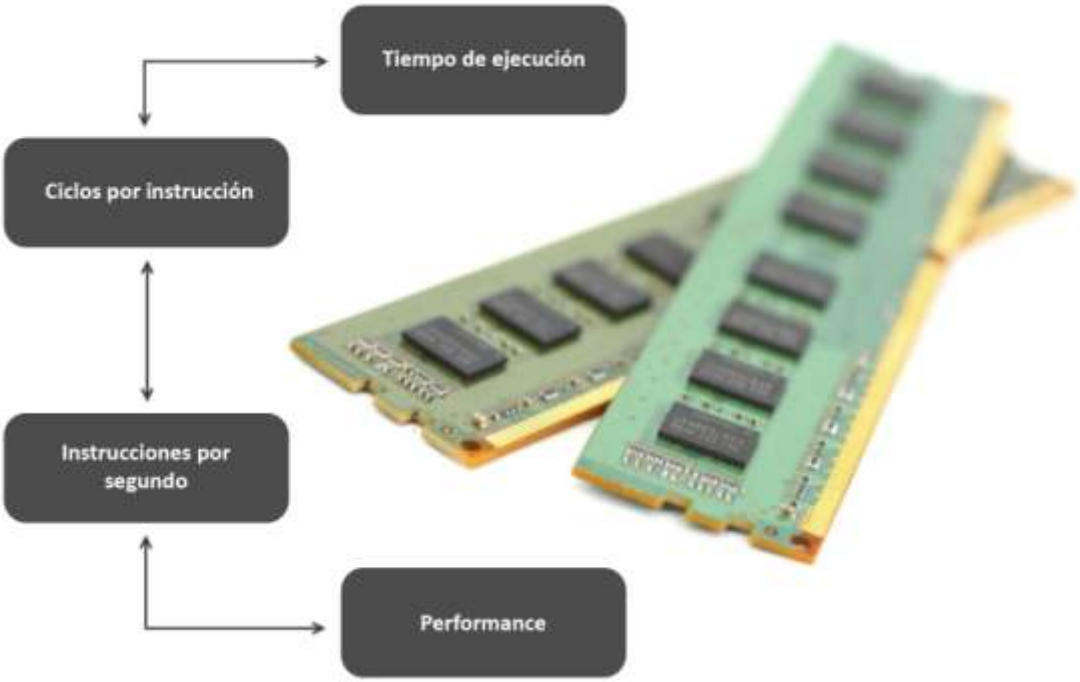
Para comparar el rendimiento de los computadores, hasta el año 2000, se usaba como medida estándar el número de veces más rápido que el VAX-11/780. El VAX-11/780 fue la primera máquina de la familia VAX. Ejecutaba aproximadamente 1 millón de instrucciones por segundo (1 MIPS) con programas reales. El procedimiento consiste en ejecutar un conjunto de **programas de prueba** llamados *benchmarks* que implementan algoritmos ya definidos, y comparar sus resultados con las tablas de tiempos ya calculadas para el VAX.

En la actualidad, en el *benchmark* estándar del mercado SPEC CPU2006, así como en su versión anterior CPU2000, se usa como máquina de referencia un servidor diseñado en 1997, llamado Sun Ultra Enterprise 2, con un procesador UltraSPARC II a 296 MHz.

Definiciones y fórmulas

En el elemento interactivo siguiente, podrás explorar las definiciones de las medidas de rendimiento más importantes y las **fórmulas** asociadas a cada medida, para saber cómo calcularlas.

Al final de este tema debemos ser expertos conocedores de estas medidas y ser capaces de calcularlas.



Tiempo de ejecución
Definición: • Tiempo que el usuario tarda en recibir una respuesta del sistema desde que ejecuta un programa, hasta que recibe el resultado por la pantalla o interfaz de usuario (acrónimo: Tej). • Tiempo que el procesador tarda en ejecutar un programa (acrónimo: T_{CPU}). $$\mathbf{T_{CPU}} = \mathbf{N} \cdot \mathbf{CPI} \cdot \mathbf{t}$$ $$\mathbf{T_{CPU}} = \mathbf{t} \cdot \sum_{i=1}^{n} (\mathbf{CPI_i} \cdot \mathbf{I_i})$$ * Unidad: segundos Leyenda: • N → Recuento de instrucciones=instrucciones/programa. • T → Período de reloj. • CPI → Ciclos por instrucción. • CPI_i → Ciclos por instrucción de tipo i. • I_i → Nº de Instrucciones de tipo i. • n → Nº de tipos de instrucciones.

Ciclos por instrucción

Definición:

- Número de ciclos de instrucción que transcurren en un segundo (acrónimo: CPI).

$$\mathbf{CPI} = \frac{\boldsymbol{C}}{\boldsymbol{N}}$$

$$\mathbf{CPI} = \sum_{i=1}^{n} (\mathbf{CPI_i} \cdot \mathbf{F_i})$$

$$\mathbf{PI} = \frac{(\mathbf{Tej} \cdot \mathbf{F})}{\mathbf{N}}$$

* Unidad: ciclos/instrucción

Leyenda:

- C → Número ciclos / programa.
- N → Recuento de instrucciones=instrucciones/ programa.
- F_i → Frecuencia (% 0..1) de aparición de instrucciones de tipo i.
- F_i → Frecuencia de Reloj (Hz).

Instrucciones por segundo

Definición:

- Millones de Instrucciones ejecutadas en un segundo (acrónimo MIPS).

$$\mathbf{MIPS} = \frac{\mathbf{N}}{\mathbf{Tej}}$$

$$\mathbf{MIPS} = \frac{\mathbf{F\ (MHz)}}{\mathbf{CPI}}$$

* Unidad: MIPS = 106 instr./seg

Leyenda:

- N → Recuento de instrucciones=instrucciones/ programa
- Tej → Tiempo de ejecución.
- F(MHZ) → Frecuencia de reloj medida en MHz.
- CPI → Ciclos por instrucción.

Performance

Definición:

- Número de veces más rápido que el UltraSPARC II (acrónimo: performance (X) = P(X)).
- X es n veces más rápido que Y (acrónimo: n = performance (X/Y)).

$$\mathbf{P(X)} = \frac{\mathbf{Tej\ (UltraSPARC\ II)}}{\mathbf{Tej\ (X)}}$$

* Unidad: Performance de X. X es P(X) veces más rápido que el UltraSPARC II.

$$\mathbf{n} = \frac{\mathbf{Tej\ (Y)}}{\mathbf{Tej\ (X)}} = \frac{\mathbf{P(X)}}{\mathbf{P(Y)}}$$

* Unidad: X es n veces más rápido que Y

Leyenda:

- Tej (X) → Tiempo de ejecución X

Ejemplo de cálculo de medidas de rendimiento

Tras ejecutar un programa de prueba en un Intel Pentium (***) (Intel, 2016) de 166 MHz, se obtuvo un tiempo de ejecución de **6 segundos** y la siguiente distribución de cada tipo de instrucciones:

Tipo de instrucción	Frecuencia de uso	Ciclos de reloj
Operaciones Aritmético-lógicas	43%	1
Carga desde Memoria	21%	2
Almacenamiento en Memoria	12%	2
Instrucciones de salto	24%	2

A continuación, se presenta el **cálculo** del CPI (Ciclos Por Instrucción), MIPS (del inglés Millions of Instructions Per Second) y el recuento total de instrucciones:

CPI

Como los datos que nos dan incluyen la distribución de instrucciones (en porcentaje) de cada tipo, cuántos ciclos dura cada tipo de instrucción, usaremos la fórmula basada en tipos de instrucción:

$$\mathbf{CPI} = \sum_{i=1}^{n} (\mathbf{CPI_i} \cdot \mathbf{F_i}) = \mathbf{0,43} \cdot \mathbf{1} + \mathbf{0,21} \cdot \mathbf{2} + \mathbf{0,12} \cdot \mathbf{2} + \mathbf{0,24} \cdot \mathbf{2} = \frac{\mathbf{1,57\ ciclos}}{\mathbf{instrucción}}$$

CPI = 1,57 CPI

MIPS

Para el cálculo de MIPS, usaremos la fórmula basada en la frecuencia del procesador y el CPI que acabamos de calcular.

$$\mathbf{MIPS} = \frac{\mathbf{F\ (MHz)}}{\mathbf{CPI}} = \frac{\mathbf{166}}{\mathbf{1,57}} = \frac{\mathbf{105,76\ Millones\ de\ Instrucciones}}{\mathbf{segundo}}$$

MIPS = 105,76

Recuento total de instrucciones

Para calcular el número total de instrucciones, tendremos que basarnos en los datos ya calculados y aplicar alguna fórmula de las que ya conocemos.

Por ejemplo, podemos usar la fórmula del tiempo de ejecución, que es un dato que sí conocemos (6 segundos): $\mathrm{TCPU} = \mathrm{N} \cdot \mathrm{CPI} \cdot \mathrm{t}$

Despejamos N (recuento de instrucciones): $\mathrm{N} = \frac{\mathrm{TCPU}}{(\mathrm{CPI} \cdot \mathrm{t})}$

No sabemos el ciclo de reloj t, pero conocemos la frecuencia de reloj $\mathrm{F} = 166\ \mathrm{MHz} = 166 \cdot 106\ \mathrm{Hz}$

Por tanto el ciclo de reloj es: $\mathrm{t} = \frac{1}{(166 \cdot 106)}$, $\mathrm{N} = \frac{6 \cdot (166 \cdot 106)}{1,57} = 634,39$ millones de instrucciones.

N= 634,39 M instr.

¿Tú lo habrías calculado de otra forma?

Tiempo de ejecución, MIPS y CPI

Ya sabemos que el tiempo de ejecución es la única medida **fiable**, pero hay otras medidas íntimamente relacionadas como son los MIPS (del inglés Millions of Instructions Per Second) y los PI (Ciclos Por Instrucción).

¿Podemos fiarnos de los MIPS para comparar el rendimiento de dos sistemas? Aparentemente los MIPS son una buena medida, pues cuántas más instrucciones ejecutemos, mejor.

Pero fijémonos en una **cuestión**: puede que, al implementar un mismo algoritmo en dos arquitecturas distintas, el programa resultante tenga muchas más instrucciones en el lenguaje máquina que usa una computadora que la otra.

¿Cómo es posible? El número de instrucciones dependerá del **juego de instrucciones**. Las arquitecturas RISC (Reduced Instruction Set Computer) tienen un juego reducido de instrucciones mucho más genéricas que las arquitecturas CISC (Complex Instruction Set Computing). Por tanto, en las arquitecturas RISC necesitaremos más instrucciones (aunque duran menos tiempo) para llevar a cabo el mismo programa.

Esto implica que aunque ejecutemos **muchas más instrucciones** por segundo en una arquitectura RISC, puede que no ejecutemos el programa en menos tiempo. Es decir, dependemos del CPI.

Las instrucciones en las arquitecturas CISC tienen un CPI mucho mayor que las arquitecturas RISC, pues las instrucciones RISC son mucho más sencillas y genéricas. Ahora bien, para hacer la misma operación, por **ejemplo**, multiplicar o calcular la raíz cuadrada, necesitamos varias instrucciones en RISC, y una sola en CISC.

Como conclusión, encontramos que el rendimiento al comparar dos máquinas, depende de la **frecuencia del procesador** (¿cuánto dura un ciclo?), y del **número de ciclos** ejecutados al finalizar el programa.

Debemos, por tanto, ejecutar el programa en las dos máquinas y **comparar** los tiempos de ejecución.

Ley de Amdahl

La Ley de Amdahl permite representar matemáticamente cómo influye una mejora sobre un componente o varios (que se utilizan durante un porcentaje del tiempo de ejecución) en la mejora global del rendimiento de una computadora.

Si mejoramos **x veces** un componente de un computador, el componente será x veces más rápido, es decir, hará el mismo trabajo en x veces menos de tiempo. Si ese componente se utiliza durante una **fracción de tiempo F** del tiempo total de la ejecución, ¿cuál será el tiempo de ejecución del sistema después de la mejora?

Tiempo de ejecución con mejora = tiempo de ejecución sin mejora * F / x + tiempo de ejecución sin mejora * (1-F)

Donde (1-F) representa la fracción de tiempo en la cual no hay mejora (en tanto por uno).

Si operamos con la fórmula anterior:

- Tiempo de ejecución con mejora = tiempo de ejecución sin mejora * (F / x + (1-F)).
- 1 / (F / x + (1-F)) = tiempo de ejecución sin mejora / tiempo de ejecución con mejora = speed-up.
- Speed-up = 1 / (F / x + (1-F)).

$$\mathbf{S} = \frac{\mathbf{x}}{[\mathbf{F} + \mathbf{x}\,(\mathbf{1} - \mathbf{F})]}$$

Donde:

- S → Es la aceleración o speed-up del sistema.
- x → Es la mejora parcial aplicada durante una fracción de tiempo F.
- F → Es la fracción de tiempo durante la que se aplica la mejora x.

Ejemplo

Un arquitecto de computadores se plantea aplicar una mejora en las operaciones en coma flotante que hará que este tipo de instrucciones se realicen en un tiempo de ejecución 10 veces menor.

Se estima que el 40% de las instrucciones operan con números reales.

¿Cuál es el *speed-up* o **aceleración del sistema** al aplicar esta mejora?

Solución
Aceleración del sistema al aplicar esta mejora $\mathbf{S} = \frac{\mathbf{x}}{[\mathbf{F} + \mathbf{x}\,(\mathbf{1} - \mathbf{F})]} = \mathbf{S} = \frac{\mathbf{10}}{[\mathbf{0,4} + \mathbf{10}\,(\mathbf{1} - \mathbf{0,4})]} = \mathbf{1,56\ veces\ más\ rápido}$

Mejora	x = 10.
Frecuencia de uso	F = 0,4

Es decir, las operaciones con números reales mejoran un 1000%, pero el **sistema global** solo mejora un **56%**.

Mejoras y limitaciones de la ley de Amdahl

La ley de Amdahl también permite calcular el speed-up o aceleración de una computadora, cuando aplicamos varias mejoras a la vez en distintos componentes.

Imaginemos que aplicamos varias mejoras a la vez: cada uno de los componentes mejorados se utiliza durante una **fracción** de tiempo fi, y cada mejora la representaremos por un factor de mejora Ri.

La fórmula de speed-up o mejora global del sistema se calcula como:

$$\mathbf{S} = \frac{\mathbf{1}}{\sum_{i=1}^{n}(\frac{\mathbf{fi}}{\mathbf{Ri}})}$$

Donde:

- Ri: cantidad de mejora de la tarea i.
- fi: fracción de tiempo que se aplica la mejora i.

¡Ojo! Hay que tener en cuenta que una de las mejoras hará referencia a la fracción de tiempo en la cual no aplicamos ninguna mejora.

Cuando no hay mejora, el factor de mejora considerado será Ri = 1.

Limitación de la ley de Amdahl

La ley de Amdahl nos permite representar cuánto es la mejora global de un sistema al aplicar una mejora sobre un **componente**. Pero esta ley tiene una limitación.

El principal inconveniente de la ley de Amdahl aparece cuando introducimos varias mejoras a la vez y queremos calcular la mejora global del sistema, pues la fórmula presentada solo funciona si las mejoras aplicadas no se solapan en el tiempo. Es decir, que durante una ejecución, nunca se están aplicando dos o más mejoras a la vez.

Por ejemplo, si queremos calcular el *speed-up* de un sistema en el que mejoremos la **velocidad del procesador** (se usa el 95% del tiempo), y los accesos a memoria (el 20% de las instrucciones son de acceso a memoria), resulta que hay momentos en los que se están aplicando ambas mejoras simultáneamente, por lo que en este caso no podríamos aplicar directamente la Ley de Amdahl.

En este caso, el *speed-up* final habría que calcularlo en dos pasos. Primero calcularíamos cuánto se reduce el tiempo de ejecución con la primera mejora y después obtendríamos el *speed-up* de la segunda mejora.

Resumen

Para poder diseñar una arquitectura óptima que ofrezca el rendimiento esperado para el fin para el que se construyó, es imprescindible conocer cuáles son las **medidas de rendimiento** que me permiten medir las prestaciones globales de un sistema. Igualmente, este conocimiento me permitirá mejorar una característica concreta o el rendimiento global de una arquitectura ya diseñada o incluso ya fabricada.

La única medida de rendimiento fiable es **el tiempo de ejecución**. El tiempo de ejecución medirá el tiempo que tarda en ejecutarse un programa de prueba en dos máquinas distintas. Esta medida nos permitirá comparar el rendimiento de una máquina respecto a la otra, o de una configuración respecto de otra dentro de una misma máquina.

Los **programas de prueba** deben tener una complejidad similar a los programas reales que normalmente se ejecutarán, para que las medidas sean lo más fiables posibles. Otras medidas importantes son los **MIPS**, el **CPI**, y la **frecuencia del procesador**.

Una habilidad importante de una arquitectura de computadores es la **mejora de un sistema hardware** mediante el avance de algunos componentes. El ingeniero de computadores cuenta con un presupuesto limitado y no podrá mejorar todos los componentes. Para decidir qué parte mejorar, hay que fijarse en aquellos componentes o tareas que se **utilizan durante más tiempo**. El ingeniero debe saber calcular la **mejora global** de un sistema cuando se aplica una mejora (o varias) sobre un componente específico.

Referencias bibliográficas

- Apache Software Foundation, The (2016). Apache Open Office. Acceso web: www.openoffice.org
- Apple (2016). OS X. Acceso web: http://www.apple.com/es/osx/
- Intel Corporation (2016). Intel Processor Pentium. Acceso web: http://www.intel.es/content/www/es/es/processors/pentium/pentium-processor.html
- Marcellus D., H. (1984). Systems programming for Computers. Prentice Hall.
- Microsoft (2016). Acceso web: www.microsoft.com, https://products.office.com/es-es/home
- Newell, A. y Simon, H. (1976). Computer science as empirical inquiry: symbols and search. Communications of the ACM. pp. 113-126.
- Patterson D., y Hennessy, J. (2011). Estructura y Diseño de Computadores: Interfaz Hardware/Software. Editorial Reverté. Cuarta Edición. Traducido y adaptado de: Patterson & Hennessy, Computer Organization and Design: Hardware/Software Interface. Forth Edition. 2009. Ed. Elsevier.
- Tanembaum, A., S. (1999). Structured Computer Organization (4th. ed.). Englewood Cliffs N.J. Prentice-Hall.
- The Open Group (2016). UNIX. Acceso web: http://www.opengroup.org/subjectareas/platform/unix