Tema 0. Introducción a la arquitectura del computador

Arquitectura y Tecnología de Computadores

Curso 2014–2015

En este tema se presenta una introducción a la arquitectura del computador a partir de los conceptos vistos en la asignatura de Fundamentos de Computadores y Redes de primer curso. Este tema sentará las bases para entender el diseño de las arquitecturas de computadores modernas.

1. Estructura básica del computador

Un computador se define como una máquina que es capaz de recibir información de entrada, procesarla bajo control de un programa y generar información a través de medios de salida, todo ello sin intervención de un operador humano.

Aunque habitualmente la imagen de un computador es una máquina con diferentes periféricos conectados como un teclado, un ratón o un monitor, la realidad es que hoy en día existen computadores de muchos tipos que van desde el típico ordenador personal o portátil a tabletas, teléfonos móviles e incluso sistemas empotrados como los que se encuentran en vehículos y electrodomésticos.

La construcción de un computador capaz de ejecutar programas plantea una serie de necesidades:

- Almacenar el programa a ejecutar. El programa a ejecutar debe estar almacenado en algún tipo de contenedor dentro del computador. También debe ser posible eliminarlo cuando ya no sea necesario.
- Almacenar los datos del programa. Los datos de entrada y salida del programa deben almacenarse (y eliminarse) en un contenedor dentro del computador.
- Mecanismo de lectura y ejecución del programa. Con el programa y los datos almacenados dentro del computador se pueden ejecutar los pasos que el primero define utilizando los datos de entrada, generando los datos de salida.
- Mecanismo de entrada/salida. Tanto el programa a ejecutar como los datos con los que trabaja deben ser introducidos al computador de alguna forma, ya sea por un usuario o por otro sistema externo. Igualmente, los datos de salida deben ser mostrados al usuario.

Tal como ocurre en otros campos de la ingeniería, el diseño de un computador debe seguir unos requisitos y cumplir una serie de restricciones de rendimiento, coste y consumo. Por tanto, el diseño de un computador está guiado habitualmente por soluciones de compromiso entre estos requisitos. Aunque existen varios posibles diseños para los computadores, todos ellos tienen en común tres componentes: la memoria, la CPU y los periféricos. La memoria constituye el contenedor donde se almacenan tanto los programas como los datos que utilizan, la CPU lee dichos programas y los ejecuta, y los periféricos muestran los datos de salida al usuario.

El diseño de computador más habitual es el basado en la arquitectura von Neumann, mostrada en la figura 1. En ella se identifican cada uno de los elementos anteriormente mencionados: memoria, CPU y periféricos de E/S. En la CPU se identifican dos elementos:

- Unidad de control: interpreta las instrucciones del programa y las ejecuta.
- Unidad aritmético-lógica (ALU): se utiliza para operar sobre datos binarios.

Existen también una serie de buses que interconectan dichos elementos. Los buses son canales de comunicación compartidos:

- Bus de direcciones. A través de este bus la CPU indica a la memoria o a los periféricos la localización de la información a leer o escribir.

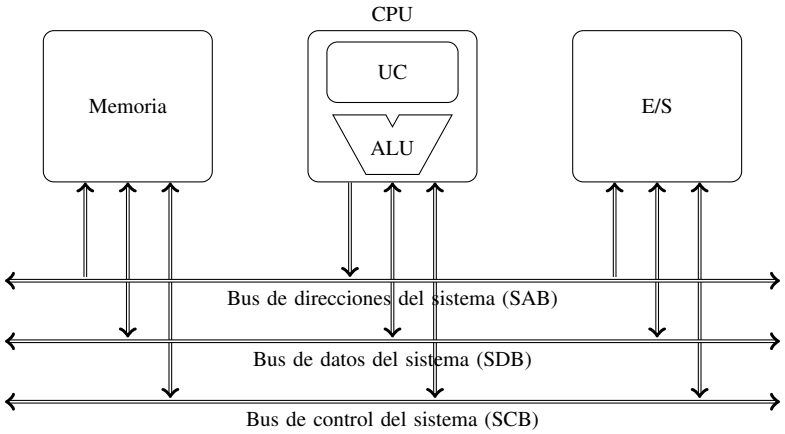

Figura 1: Arquitectura von Neumann

- Bus de datos. Sobre este bus se transmite la información entre los diferentes elementos del computador.
- Bus de control. Contiene líneas que gestionan el trasiego de información dentro del computador.

2. Máquina multi-nivel

Un desafío que deben afrontar los computadores de hoy en día es la «distancia conceptual» que existe entre el lenguaje que entiende el computador —el lenguaje de la máquina, codificado en una secuencia de bits— y el lenguaje que utilizan los programadores para implementar sus algoritmos. Esta distancia es cada vez mayor debido a dos factores:

- Al programador le interesa utilizar un lenguaje próximo al lenguaje natural para describir las soluciones a los problemas a resolver, con lo que se simplifica el desarrollo de los programas.
- Interesa que el lenguaje del computador sea lo más sencillo posible para poder utilizar el menor número de recursos hardware posible y al mismo tiempo que sea altamente eficiente.

Para lidiar con este problema es común definir un nuevo conjunto de instrucciones más próximo al lenguaje natural, L1, a partir del lenguaje que utiliza la máquina, L0. De esta forma, el lenguaje L1 se define a partir de las instrucciones disponibles en el lenguaje L0. Este nuevo lenguaje da lugar a una máquina virtual, M1, construida a partir de la máquina física, M0, que puede ser utilizada por los programadores para desarrollar programas en el lenguaje L1, más cercano al lenguaje natural.

Para ejecutar programas escritos en el lenguaje L1 sobre la máquina M0 (que utiliza el lenguaje L0), son posibles dos técnicas no excluyentes:

- Traducción: Antes de ejecutar el programa L1, este debe traducirse completamente a instrucciones en el lenguaje L0.
- Interpretación: El programa L1 sirve como entrada a un programa escrito en L0 (intérprete) que se encarga de ejecutar las instrucciones L1 sobre la máquina M0.

Si bien la definición de un nuevo lenguaje permite simplificar el desarrollo de programas por parte de los programadores, la distancia entre los lenguajes L1 y L0 no debe ser excesiva para poder realizar la traducción y/o interpretación, con lo que el lenguaje L1 todavía sigue estando lejos del lenguaje natural. Para solucionar este problema, se pueden definir más niveles para aproximarse progresivamente al lenguaje natural. Hoy en día, los computadores suelen definir un conjunto de niveles comunes:

- Nivel de lógica digital (nivel 0): Formado por puertas lógicas, constituye el hardware del computador.
- Nivel de microarquitectura (nivel 1): Este nivel está formado por el camino de datos. En algunas máquinas las operaciones sobre el camino de datos se controlan con un microprograma. En este caso, el microprograma es el intérprete de las instrucciones del nivel 2. La tendencia hoy en día es a eliminar este microprograma y controlarlas directamente por hardware, o al menos, seguir una aproximación híbrida donde se controlan directamente por hardware las operaciones más sencillas (habitualmente las más comunes) y a través de un microprograma las más complejas.

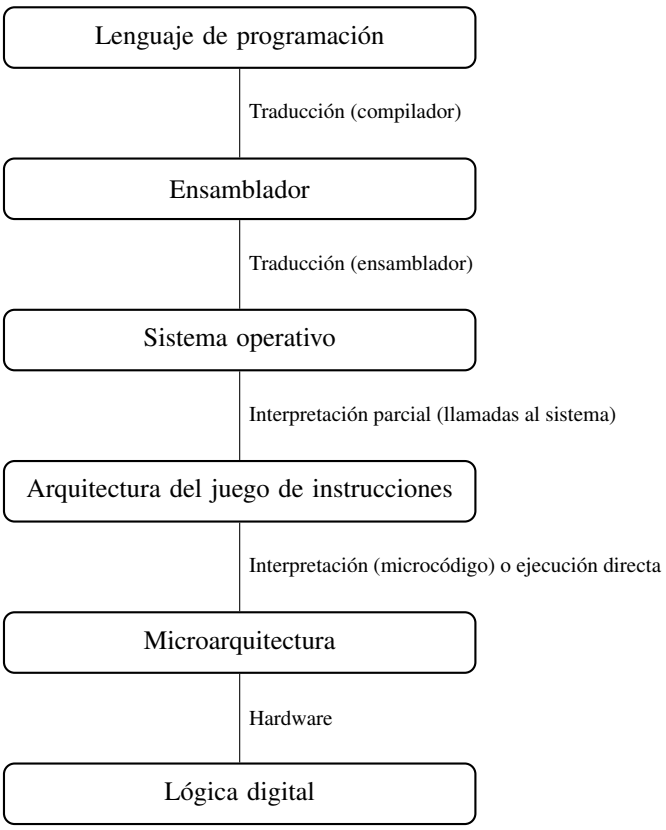

Figura 2: Niveles virtuales del computador actual

- Nivel de arquitectura del juego de instrucciones (nivel 2): Especificación del fabricante del lenguaje que entiende el procesador, los registros disponibles, los modos de direccionamiento, etc.
- Nivel de sistema operativo (nivel 3): Es un nivel híbrido donde muchas instrucciones son del nivel 2, y otras se interpretan (llamadas a servicios del sistema operativo).
- Nivel de lenguaje ensamblador (nivel 4): Es el nivel más bajo en el que pueden desarrollarse los programas de usuario. Es el primer lenguaje simbólico y habitualmente es traducido (compilado) al nivel 3.
- Nivel 5 y superiores: Son niveles que definen lenguajes de alto nivel, por ejemplo C o C++.

La figura 2 resume los niveles que habitualmente se definen en el computador de hoy en día y cuál es el método más habitual para ejecutar los programas definidos en una máquina virtual sobre la máquina inmediatamente inferior.

3. Arquitectura del computador

El diseño de un computador supone definir cuál es su arquitectura, que en gran medida se refiere a la arquitectura de la CPU que lo gobierna. Esta arquitectura comprende aspectos tales como qué elementos componen el computador y cuál es su comportamiento e interacciones, cómo se interconectan estos elementos o qué mecanismos de entrada salida utiliza el computador para comunicarse.

Existen dos grandes arquitecturas que pueden seguirse al diseñar un computador. La arquitectura von Neumann, vista anteriormente, y la arquitectura Harvard, donde existen dos memorias, en una de las cuales se almacenan los programas y en la otra los datos con los que trabajan.

Hoy en día, la arquitectura de una CPU, y por extensión la del computador, suele referirse a la arquitectura del juego de instrucciones (o ISA de su acrónimo en inglés). La definición de la **arquitectura del juego de instrucciones** de una CPU es la decisión más crítica cuando se diseña una CPU, ya que determina todos aquellos aspectos visibles al programador[1]. Incluye, entre otras cosas, el tipo de datos que maneja la CPU (enteros, números reales, cadenas, etc.), los registros que están a disposición del programador, el juego de instrucciones disponible, o cuál es el formato de las direcciones y los modos de direccionamiento.

Las arquitecturas del juego de instrucciones se han clasificado históricamente en dos grandes grupos: las arquitecturas RISC (Reduced Instruction Set Computer) y CISC (Complex Instruction Set Computer). Las arquitecturas RISC definen

[1] Programador a bajo nivel, desarrollador de compiladores o programador de sistemas.

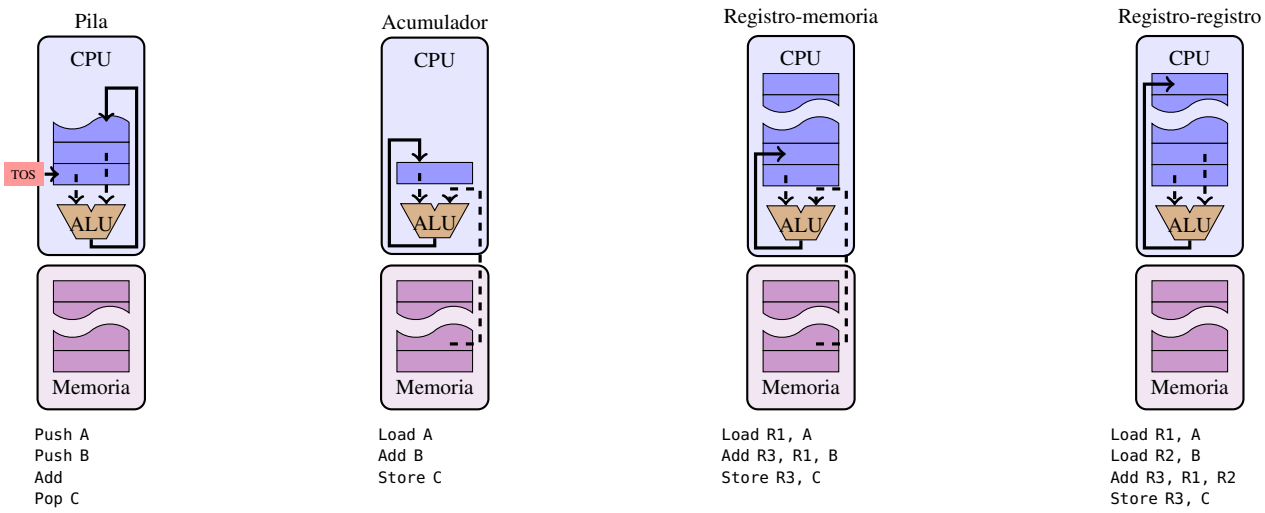

Figura 3: Tipos de modelo de máquina

instrucciones muy simples que pueden ejecutarse rápidamente. Con esto se consigue que el hardware necesario sea mínimo. Algunos ejemplos de este tipo de arquitecturas son: ARM (muy extendida en dispositivos móviles), MIPS, IA-64 o PowerPC. En cambio, las arquitecturas CISC definen instrucciones complejas que pueden realizar operaciones de alto nivel, facilitando la labor del programador. Ejemplos de arquitecturas CISC incluyen: x86 o Motorola 68 000.

La definición del juego de instrucciones de una CPU puede responder a cuatro modelos de máquina distintos, ilustrados en la figura 3.

- Pila. En una máquina de pila, la CPU dispone de un conjunto de registros a modo de pila y de un registro especial denominado cima de la pila (`TOS`, *top of stack*). Los operandos de todas las instrucciones que puede ejecutar la CPU son implícitos y se obtienen de la pila, empezando por el apuntado por el registro `TOS`. El resultado de la operación se almacena también en la cima de la pila. Por ejemplo, una operación de tipo suma podría tener la siguiente forma:

  ```
  push A
  push B
  add
  pop C
  ```

  Inicialmente se cargan en la pila los valores de las variables `A` y `B`, tras lo cual se llama a la instrucción de suma. Esta instrucción extrae de la pila los dos últimos valores, los suma y almacena el resultado en la misma pila. La instrucción `pop` extrae el valor apuntado por el registro `TOS` y lo almacena en la variable `C`. La ventaja de una CPU basada en una modelo de máquina de pila es un menor de número de bits necesario para codificar las instrucciones.

- Acumulador. En una máquina de tipo acumulador existe un registro especial, denominado precisamente registro acumulador, que se utiliza implícitamente como operando. El resultado de una operación se almacena en el registro acumulador. De esta forma, una operación de suma podría tener la siguiente forma:

  ```
  load A
  add B
  store C
  ```

  La primera instrucción carga en el registro acumulador el valor de la variable `A`. A continuación, se puede realizar la operación de suma, indicando la variable cuyo valor va a sumarse con el valor del registro acumulador. El resultado almacenado en el registro acumulador se guarda en la variable `C` con la última instrucción.

- Registro-memoria. En una máquina de tipo registro-memoria un operando debe estar almacenado en uno de los registros del fichero de registros que incorpora la CPU y el otro directamente en memoria. El resultado de la operación se almacena en otro de los registros del fichero de registros. Bajo estas condiciones, una operación de suma podría tener la siguiente forma:

  ```
  load r1, A
  add r3, r1, B
  store r3, C
  ```

  La primera instrucción carga en el registro `r1` el valor de la variable `A`. A continuación, se realiza la suma entre el valor almacenado en este registro y la variable `B` almacenada en memoria. El resultado se almacena en el registro `r3`

del fichero de registros. Finalmente, el resultado de la suma puede almacenarse en memoria en la variable `C` desde el registro `r3`.

- Registro-registro o carga/almacenamiento. En una máquina de tipo registro-registro todos los operandos con los que trabaja una instrucción deben estar en el fichero de registros de la CPU. La misma operación de suma tendría la siguiente forma en una máquina de este tipo:

```
load r1, A
load r2, B
add r3, r1, r2
store r3, C
```

  Las primeras instrucciones cargan en los registros `r1` y `r2` los operandos. La instrucción de suma debe indicar explícitamente cuáles son los registros que contienen los operandos y en cuál se almacenará el resultado. Tras realizar la operación, el resultado puede almacenarse en memoria en la variable `C` desde el registro `r3`.

Por otra parte, la **microarquitectura** se refiere a cuál es la organización interna de la CPU, no visible al programador, para implementar una determinada arquitectura del juego de instrucciones. Por esta razón, es posible definir varias microarquitecturas que implementen la misma arquitectura del juego de instrucciones. Por ejemplo, Nehalem (Core i5) y AMD K10 (Athlon II) son microarquitecturas de Intel y AMD respectivamente que implementan la misma arquitectura del juego de instrucciones (x86).

La microarquitectura define aspectos tales como el número de unidades funcionales dentro de la CPU, por ejemplo el número de unidades aritmetico-lógicas, el número y tipo de cachés o el ancho de los buses de memoria. También determina el tipo de organización, dando lugar a distintos tipos CPU: segmentadas, superescalares o multinúcleo, entre otras. Estos tipos de organizaciones serán vistos a lo largo del tema dedicado a la CPU.

Tema 1. Análisis cuantitativo del rendimiento del computador

Arquitectura y Tecnología de Computadores

Curso 2014–2015

El rendimiento es un factor crucial a la hora de comparar dos computadores. En ocasiones el rendimiento está directamente relacionado con el coste, pero no siempre un computador de mayor coste ofrece un rendimiento superior a otro de menor coste. Es por tanto necesario poder evaluar el rendimiento que ofrece un computador de forma objetiva.

En este capítulo se abordará el problema de evaluar el rendimiento de un computador, al tiempo que se sentarán las bases necesarias para comprender las mejoras del rendimiento que se plantearán a lo largo del curso en los diferentes componentes del computador.

1. Concepto de rendimiento

Un computador es un sistema muy complejo en el que se interrelacionan muchos elementos. Resulta complicado delimitar el concepto de rendimiento de un computador, ya que es muy amplio, cubriendo aspectos tales como el tiempo que invierte en ejecutar una tarea, el número de tareas que ejecuta por unidad de tiempo, el consumo eléctrico, etc. Por esta razón, es habitual utilizar diferentes formas de medir el rendimiento a través de lo que se conocen como métricas de rendimiento. Una **métrica** es una unidad que cuantifica un aspecto medible de un sistema. En el caso de las métricas de rendimiento, intentan cuantificar el rendimiento de un computador, es decir, proporcionan un número de tal forma que pueden compararse los valores de dicha métrica para varios computadores.

Comúnmente se utilizan dos métricas para evaluar el rendimiento de un computador: el tiempo de respuesta y la productividad. El **tiempo de respuesta** se refiera al tiempo que invierte el computador en realizar una tarea. Por ejemplo, el tiempo desde que se inicia la ejecución de un programa y este devuelve el resultado, el tiempo que tarda un gestor de bases de datos en proporcionar los resultados a una consulta, etc. El rendimiento entonces se expresa como la inversa del tiempo de respuesta, e indica el número de veces que podría llevarse a cabo la misma tarea por unidad de tiempo. Es decir, un computador tendrá un rendimiento mayor cuanto menor sea el tiempo de respuesta en la ejecución de la tarea.

$$\text{Rendimiento} = \frac{1}{\text{T. respuesta}}$$

Por ejemplo, si el tiempo de respuesta de una tarea es de $1\,\text{ms}$, podrían llevarse a cabo $(1/0.001\,\text{s}) = 1000$ tareas idénticas por segundo.

En ocasiones se habla indistintamente de tiempo de respuesta y de velocidad del computador, de tal forma que un computador rápido es aquel que tiene un tiempo de respuesta bajo. En este caso se equipara velocidad y rendimiento del computador. A modo de ejemplo, supongamos que tenemos un computador A que es capaz de completar una tarea en 20 segundos, mientras que otro computador B es capaz de completar la misma tarea en 30 segundos. ¿Cuántas veces el computador A es más rápido que el computador B, es decir, cuál es el factor por el que el rendimiento del computador A se multiplica respecto al del computador B?

$$\frac{\text{Rendimiento}_\text{A}}{\text{Rendimiento}_\text{B}} = \frac{\text{T. respuesta}_\text{B}}{\text{T. respuesta}_\text{A}} = \frac{30}{20} = 1.5$$

Si consideramos la velocidad del computador como la inversa del tiempo de respuesta, también es posible hablar de aceleración de un computador, referido a una mejora del rendimiento. Concretamente, la aceleración se refiere al factor por el que se mejora la velocidad, que para el caso anterior es 1.5. Se dice que el computador A tiene una aceleración (*speedup*) de 1.5, o lo que es lo mismo, es un $50\,\%$ más rápido que el computador B:

$$\frac{\text{T. respuesta}_\text{B} - \text{T. respuesta}_\text{A}}{\text{T. respuesta}_\text{A}} \times 100 = \frac{30 - 20}{20} \times 100 = 50\,\%$$

Otra de las métricas habitualmente utilizadas cuando se mide el rendimiento del computador es la productidad. La **productividad** indica el número de tareas completadas por unidad de tiempo. Por ejemplo, el número de tareas que ejecuta por unidad de tiempo, el número de peticiones servidas por una base de datos en un segundo, etc. Un computador tendrá un rendimiento mayor cuanto mayor sea su productividad.

La productividad es también un concepto que se puede medir de varias formas. Por ejemplo, la productividad de un sistema de gestión de bases de datos puede medirse según el número de consultas ejecutadas por unidad de tiempo o el número de usuarios concurrentes que pueden servirse.

En el caso de un computador que no puede ejecutar concurrentemente dos o más tareas, la productividad en la ejecución de tareas puede calcularse como la inversa del tiempo de respuesta. En el ejemplo anterior, asumiendo que ambos computadores disponen de una única CPU, la productividad (ρ) de ambos computadores, medida en tareas por segundo, puede calcularse como la inversa del tiempo de respuesta, de tal forma que:

$$\rho_{\text{A}} = \frac{1}{\text{T. respuesta}_{\text{A}}} = \frac{1}{20} = 0.05 \text{ tareas/s}$$

$$\rho_{\text{B}} = \frac{1}{\text{T. respuesta}_{\text{B}}} = \frac{1}{30} = 0.0\widehat{3} \text{ tareas/s}$$

Se pueden plantear mejoras en el rendimiento del computador que afecten a la vez al tiempo de respuesta y a la productividad, o bien que únicamente mejoren esta última. Por ejemplo, si somos capaces de reducir el tiempo de respuesta de la tarea en el computador A de tal forma que ahora tarda 15 segundos en completarse, la productividad se incrementa hasta las 4 tareas por minuto (o $0.0\widehat{6}$ tareas/s). Sin embargo, si se mantiene el tiempo de respuesta de la tarea pero se añade un segundo procesador para poder ejecutar dos tareas en paralelo, el tiempo de respuesta se mantiene en 20 segundos, pero la productividad se multiplica por 2, ya que en un minuto se ejecutarían 6 tareas cuando antes se ejecutaban 3.

A partir de este ejemplo se pueden deducir cuáles son los factores que se pueden mejorar para aumentar el rendimiento. Por un lado, se puede disminuir el tiempo de respuesta, que implicará también un aumento de la productividad. Para conseguir este objetivo, es necesario incorporar mejoras tecnológicas u organizativas dentro del computador. Por otro lado, se puede incrementar el nivel de paralelismo de los distintos componentes del computador. Esto consigue aumentar la productividad, aunque en general no consigue disminuir el tiempo de respuesta, llegando incluso a aumentarlo.

El término tiempo de respuesta suele utilizarse cuando se evalúa el rendimiento en la ejecución de tareas. No obstante, cuando se analiza el rendimiento de sistemas de memoria se utilizan más a menudo los términos **latencia** (por tiempo de respuesta), referido al tiempo que tarda el sistema de memoria en proporcionar los datos solicitados, y **ancho de banda** (por productividad), referido a la cantidad de información que puede proporcionar por unidad de tiempo.

Aunque hasta ahora se ha considerado que el tiempo de respuesta y la productividad son medidas absolutas, en la mayor parte de las ocasiones deben aproximarse a través de variables aleatorias, pues varían entre unas mediciones y otras. Esto quiere decir que dependiendo del momento de la medida y de la carga del computador en ese instante, los resultados varían. De esta forma, por ejemplo, en lugar de tomar un tiempo de respuesta fijo, se aproxima a través de una variable aleatoria, que habitualmente se considera normalmente distribuida $\sim \mathcal{N}(\mu, \sigma^2)$, categorizada por un valor medio y una desviación típica.

Por esta razón, cobra especial importancia la estadística cuando se mide el rendimiento de los computadores. Por poner un ejemplo, si se mide 10 veces (n) el tiempo de respuesta en la ejecución de un programa en un computador y se obtienen los siguientes resultados:

Medida	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
T. resp (s)	3.2	2.9	3.1	3.0	2.8	3.1	3.2	3.0	3.3	2.7

se puede calcular la media ($\overline{x}$) y la desviación típica (s) de las medidas tomadas, es decir, la media y la desviación típica muestrales:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{3.2 + 2.9 + 3.1 + 3.0 + 2.8 + 3.1 + 3.2 + 3.0 + 3.3 + 2.7}{10} = 3.03 \text{ s}$$

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}} = 0.189 \text{ s}$$

Para estimar la media poblacional (μ) usamos como estimador la media muestral ($\overline{x}$), y la desviación típica muestral (s) como estimador de la desviación típica poblacional (σ). Asumiendo que el tiempo de respuesta sigue una distribución normal, el área comprendida en el intervalo $[\mu - 2\sigma, \mu + 2\sigma]$ abarca aproximadamente el $95\,\%$ de todos los tiempos de respuesta para el programa, o lo que es lo mismo, el tiempo de respuesta del programa se encuentra con una probabilidad del $95\,\%$ en ese intervalo[1], en este ejemplo $[2.62, 3.38]$. Esto se aprecia en la función de densidad de probabilidad del tiempo de respuesta en la figura 1.

[1] En realidad, dado que tenemos un número de medidas bajo (< 30) y que desconocemos la desviación típica poblacional, habría que utilizar una distribución t de *Student* para calcular el intervalo de confianza. Por simplicidad, siempre utilizaremos la distribución normal, con lo que el intervalo $[\mu - 2\sigma, \mu + 2\sigma]$ es una aproximación válida para una confianza del $95\,\%$.

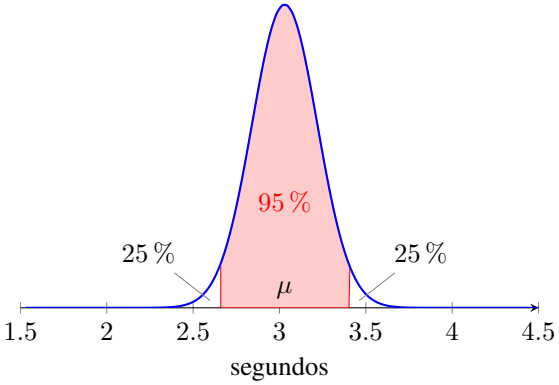

Figura 1: Función de densidad de probabilidad del tiempo de respuesta

2. Ley de Amdahl

Como se ha estudiado en la asignatura de Fundamentos de Computadores y Redes de primer curso, el computador está formado por varios componentes que trabajan conjuntamente para ejecutar los programas (CPU, memoria, sistema de interconexión y sistema de entrada/salida). Las mejoras en el rendimiento del computador se consiguen introduciendo mejoras en los distintos componentes que lo forman. No obstante, una mejora en el rendimiento de un componente en un factor p no incrementa el rendimiento de todo el computador en ese mismo factor.

La Ley de Amdahl determina que:

> *La aceleración o ganancia obtenida (A) en el rendimiento de un sistema completo debido a la mejora de uno de sus componentes está limitada por la fracción de tiempo que se utiliza dicho componente.*

El tiempo de respuesta del computador utilizando el componente mejorado se calcula a través de la suma de dos factores: el tiempo de respuesta debido a los componentes que no han sido mejorados, que permanece constante, y el tiempo empleado utilizando el componente mejorado, que se ve reducido en función de la mejora aplicada:

$$\text{T. respuesta}_{\text{mejorado}} = \text{T. respuesta}_{\text{original}} \times \left((1 - \text{Fracción}_{\text{mejorada}}) + \frac{\text{Fracción}_{\text{mejorada}}}{\text{Acelerac.}_{\text{mejorada}}} \right)$$

La fórmula de Amdahl puede deducirse a partir de la aceleración del rendimiento obtenida utilizando el componente mejorado respecto al rendimiento con el componente original:

$$A = \frac{\text{Rendimiento}_{\text{mejorado}}}{\text{Rendimiento}_{\text{original}}} = \frac{\text{T. respuesta}_{\text{original}}}{\text{T. respuesta}_{\text{mejorado}}}$$

Sustituyendo, tenemos que:

$$A = \frac{1}{(1 - \text{Fracción}_{\text{mejorada}}) + \frac{\text{Fracción}_{\text{mejorada}}}{\text{Acelerac.}_{\text{mejorada}}}}$$

Por ejemplo, supongamos que tenemos un computador que tarda en ejecutar un programa 100 segundos. De este tiempo, 20 segundos son debidos a la ejecución por parte de la CPU, 35 a los accesos a memoria, 40 a operaciones de entrada salida sobre el disco duro, mientras que el resto se debe a la espera por los diferentes mecanismos de interconexión dentro del computador. ¿Cuál sería la aceleración en la ejecución del programa que se obtendría si se sustituye la CPU por otra el triple de rápida? ¿Y si se sustituye el disco duro por uno el doble de rápido? ¿Cuáles serían los nuevos tiempos de ejecución?

Inicialmente, se calcula la fracción de tiempo correspondiente a los componentes a mejorar:

$$\text{Fracción}_{\text{CPU}} = \frac{20}{100} = 0.2$$

$$\text{Fracción}_{\text{disco}} = \frac{40}{100} = 0.4$$

Para el caso de sustituir la CPU:

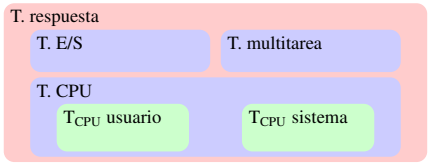

Figura 2: Factores que afectan al tiempo de respuesta

$$A = \frac{1}{(1-0.2)+\frac{0.2}{3})} \approx 1.15 \Rightarrow 15\,\% \text{ mejora}$$

$$\text{T. respuesta}_{\text{mejorado}} = \frac{\text{T. respuesta}_{\text{original}}}{A} \approx \frac{100}{1.15} = 86.96 \text{ segundos}$$

En el caso de que sustituya el disco:

$$A = \frac{1}{(1-0.4)+\frac{0.4}{2})} = 1.25 \Rightarrow 25\,\% \text{ mejora}$$

$$\text{T. respuesta}_{\text{mejorado}} = \frac{\text{T. respuesta}_{\text{original}}}{A} = \frac{100}{1.25} = 80 \text{ segundos}$$

Como se puede observar, la mejora de rendimiento sustituyendo el disco por uno el doble de rápido es mayor que la obtenida sustituyendo la CPU por una el triple de rápida. La razón es que el disco se emplea en el ejemplo durante una fracción de tiempo mucho mayor.

3. Evaluación del rendimiento de la CPU

Para comparar el rendimiento de dos CPU es necesario definir métricas que proporcionen una indicación del rendimiento del computador. El tiempo de respuesta o la productividad son dos formas distintas de medir el rendimiento, como se ha visto. Sin embargo, estas métricas hacen que la comparación del rendimiento de dos CPU sea en la práctica complicada.

El tiempo de respuesta de un programa incluye el tiempo que el programa está realmente ejecutándose, el tiempo que está esperando por operaciones de entrada/salida, etc. Por esta razón, el tiempo de respuesta suele ser muy variable, especialmente en los sistemas operativos multitarea, donde el sistema operativo no necesariamente prioriza que las tareas tengan un tiempo de respuesta bajo, con lo que una tarea podría estar bloqueada esperando a que otra tarea termine.

Algo parecido ocurre con las medidas de productividad, tales como los millones de instrucciones por segundo (MIPS), ya que estas medidas no son comparables en dos CPU que no implementan el mismo juego de instrucciones.

Es por tanto necesario considerar medidas de rendimiento que sean extrapolables a diferentes computadores de tal forma que se puedan comparar.

3.1. Tiempo de CPU

El tiempo de respuesta de un programa tal como se ha considerado hasta ahora indica el tiempo que transcurre desde que comienza el programa hasta que finaliza. Esto incluye el tiempo de ejecución del programa, el tiempo de espera por operaciones de entrada/salida, la ejecución del sistema operativo y otras tareas, etc[2]. Por este motivo, el tiempo de respuesta de una tarea es altamente variable. En la figura 2 se muestran los factores que influyen en el tiempo de respuesta.

La parte del tiempo de respuesta que comprende la ejecución del programa se denomina tiempo de CPU. Este tiempo no incluye el tiempo de espera por la entrada/salida ni la ejecución de otros programas en paralelo en un sistema operativo multitarea. El tiempo de CPU puede a su vez dividirse en:

- Tiempo de CPU de usuario. Es el tiempo de CPU dedicado a la ejecución del código del programa. Este tiempo puede utilizarse como una referencia del rendimiento que ofrece la CPU.
- Tiempo de CPU de sistema. Es el tiempo de CPU invertido en la ejecución del sistema operativo durante la ejecución de todo el programa.

En los sistemas operativos UNIX existe un comando que nos muestra el tiempo de respuesta y el tiempo de CPU de un programa. Se trata del comando |time|, que puede ser invocado de la siguiente forma:

[2]En algunos textos se utiliza *tiempo de ejecución* para referirse al tiempo de respuesta y en otros para referirse al tiempo de CPU. Para evitar confusiones, utilizaremos tiempo de respuesta y tiempo de CPU para referirnos al tiempo que tarda en finalizar el programa y al tiempo efectivo de ejecución por parte de la CPU, respectivamente.

```
$> time ./programa

real  0m2.822s
user  0m2.760s
sys   0m0.008s
```

El primer tiempo (|real|) se corresponde con el tiempo total transcurrido, es decir, el tiempo de respuesta. En este caso, el programa termina 2.822 segundos tras ser lanzado. De este tiempo, únicamente corresponde a tiempo de CPU la suma de los tiempos de CPU de usuario (|user|) y de sistema (|sys|). Así, $2.76 + 0.008 = 2.768$ segundos se corresponden con el tiempo de CPU del programa, es decir, un $2.768/2.822 \times 100 = 98.09\,\%$. Esto quiere decir que esta tarea es intensiva en CPU, ya que el tiempo invertido para entrada/salida es mínimo.

En un programa monohilo el tiempo de CPU depende a su vez de varios factores. El primer factor importante a considerar es el período de reloj (T), que es el inverso de la frecuencia (f). De esta forma puede hablarse indistintamente de período o frecuencia de reloj.

$$f = T^{-1} = \frac{1}{T}$$

Entonces, puede definirse el tiempo de CPU de un programa en relación a la frecuencia o el período de la señal de reloj como:

$$\mathrm{T_{CPU}} = \frac{\text{Ciclos de CPU del programa}}{f} = \text{Ciclos de CPU del programa} \times T$$

Los Ciclos Por Instrucción (CPI) indican el número de ciclos que son necesarios para completar cada instrucción del programa y se pueden calcular como:

$$\mathrm{CPI} = \frac{\text{Ciclos de CPU del programa}}{\text{Instrucciones del programa}}$$

A partir de los CPI, se puede calcular el tiempo de CPU como:

$$\mathrm{T_{CPU}} = \frac{\text{Instrucciones del programa} \times \mathrm{CPI}}{f} = \frac{\text{Instrucciones}}{\text{Programa}} \times \mathrm{CPI} \times T$$

A esta fórmula se la conoce como la ley de hierro (*Iron Law*) del rendimiento de una CPU. Como se puede observar en la ecuación anterior, el tiempo de CPU final depende de tres factores relacionados:

- El número de instrucciones del programa. Este número depende básicamente del nivel de abstracción al que el programador desarrolló el programa. Los lenguajes de programación de alto nivel como C++ confían en los compiladores para generar programas compactos con un número de instrucciones bajo. Por tanto, reducir el número de instrucciones por programa requiere mejorar la tecnología de compiladores para generar código optimizado.

- Los ciclos por instrucción (CPI). Dependen de la organización interna de la CPU y de la complejidad de las instrucciones a ejecutar. Las CPU CISC (*Complex Instruction Set Computer*) implementan juegos de instrucciones de complejidad elevada, con lo que se consiguen programas con un número de instrucciones bajo, lo que reduce el número de instrucciones por programa. Por contra, es más complejo ejecutar dichas instrucciones, con lo que aumentan los CPI. Las CPU RISC (*Reduced Instruction Set Computer*) o de arquitectura *load-store* implementan un juego de instrucciones muy simple, con lo que se reducen los CPI, pero implica un mayor número de instrucciones por programa.

  El principio básico enunciado por la ley de Amdahl de que la ganancia de rendimiento es proporcional a la fracción de uso del componente mejorado puede ser utilizado para reducir los CPI. Esto se traduce en que los diseñadores de juegos de instrucciones buscan reducir los CPI de las instrucciones más comúnmente utilizadas en los programas, frente a instrucciones menos utilizadas, pues su peso en el rendimiento final será menor.

- El período de reloj. Indica lo rápido que trabaja la CPU. Para reducir el ciclo de reloj, y aumentar la frecuencia de trabajo, es habitualmente necesario mejorar la tecnología de fabricación de CPU para conseguir circuitos más rápidos. No obstante, también se puede disminuir el período de reloj incluyendo mejoras organizativas como el *pipeline*, tal como se verá en el tema dedicado a la CPU.

3.2. Benchmarks

Un *benchmark* es un programa pensado para evaluar el rendimiento de un computador o de una de sus partes. Son habituales los benchmarks para evaluar el rendimiento de la CPU y del sistema de memoria.

Como se ha visto, el rendimiento de un computador no puede calcularse directamente la mayor parte de las veces debido a su complejidad, sino que debe estimarse a partir de mediciones. El resultado de estas mediciones varía en función de las condiciones a las que está sometido el computador. Un benchmark somete el computador, o la parte del mismo a

evaluar, a una determinada **carga de trabajo** para medir el rendimiento bajo la misma. Por esta razón, los resultados para distintos bechmarks pueden variar para el mismo computador.

Un benchmark debería someter al computador a una carga representativa del trabajo habitual para el que el computador está pensado, de tal forma que los resultados sean significativos. Por ejemplo, si se desea evaluar el rendimiento de un computador orientado a tareas de ofimática utilizando un benchmark, la carga que representa el benchmark debería ser parecida a la que supondrían los programas ofimáticos que se utilizarían en el día a día sobre el computador. Por otro lado, en la evaluación de una estación de trabajo dedicada al desarrollo de programas en lenguaje C debería utilizarse un benchmark con una carga representativa de compiladores, entornos integrados de desarrollo, enlazadores, etc. Por tanto, un bechmark está pensado para evaluar el rendimiento de un computador bajo unas determinadas circunstancias. Un computador puede obtener un bajo rendimiento medido con un benchmark con una determinada carga, mientras que puede obtener un rendimiento alto si se evalúa con otra carga distinta.

Hay varios tipos de benchmark dependiendo del tipo de carga que utilizan:

- Carga real. Son benchmarks que utilizan programas reales usados en el trabajo habitual del computador sobre el que se va a aplicar el benchmark. La principal ventaja es que evalúan el rendimiento bajo unas condiciones reales de utilización del computador. Su gran inconveniente es que son difíciles de reproducir, con lo que las medidas obtenidas con el benchmark pueden tener gran variabilidad.

- Carga sintética. Estos benchmarks ejecutan pequeños programas que intentan reproducir las operaciones más habituales que se desarrollan sobre los programas reales. Sin embargo, no son programas reales. Su ventaja es que son fácilmente reproducibles. Por contra, los resultados obtenidos no siempre se corresponden con el rendimiento observado en la ejecución de programas reales.

- Carga analítica. Se utilizan cuando no es posible medir directamente el rendimiento sobre el computador real, pues o bien no está disponible (no existe o no se ha adquirido) o bien no puede someterse a pruebas por tratarse de un computador que está en fase de producción. Sirven para evaluar el rendimiento esperado de un computador (o uno de sus componentes) a partir de un modelo matemático del mismo.

Existen benchmarks para estimar el rendimiento de muchas partes del computador. Los hay orientados a medir el rendimiento de la CPU, del sistema de memoria, de la interfaz gráfica, del disco duro, etc.

Tema 2. La CPU (complementos)

Arquitectura y Tecnología de Computadores

Curso 2014–2015

En este apartado se complementa el capítulo 2, dedicado a la CPU, del libro *Apuntes de organización de computadores*. En concreto, se extienden los conceptos relacionados con las mejoras de rendimiento, cubriendo los apartados *2.3 Taxonomía de Flynn* y *2.6 Procesadores multi-núcleo* de la guía docente, que no estaban tratados en el citado libro.

3. Taxonomía de Flynn

La taxonomía de Flynn es una clasificación de arquitecturas de computadores basada en el número de instrucciones concurrentes y los flujos de datos disponibles en cada arquitectura.

A partir de este criterio las arquitecturas de computadores se pueden clasificar en cuatro grandes grupos: SISD, SIMD, MISD y MIMD. A continuación se presentan las principales características de cada grupo.

SISD (*Single Instruction, Single Data*). La arquitectura SISD describe la organización de un computador secuencial que ejecuta un flujo de instrucciones único sobre un flujo de datos único.

Un ejemplo de este tipo de arquitectura son los sistemas monoprocesadores utilizados en los computadores personales a principios de la década de 1980.

SIMD (*Single Instruction, Multiple Data*). La arquitectura SIMD describe la organización de un sistema que ejecuta un único flujo de instrucciones sobre múltiples flujos de datos, es decir, que ejecuta la misma instrucción sobre varios datos a la vez.

Un ejemplo de sistema que implementa este tipo de arquitectura es la unidad de procesamiento de gráficos, o GPU, de un computador personal, ya que en el tratamiento de gráficos es habitual tener que hacer la misma operación para todos los píxeles de una imagen. También los procesadores actuales para computadores personales tienen una serie de instrucciones que se ejecutan a la vez sobre múltiples datos. Un ejemplo son las instrucciones SSE.

MISD (*Multiple Instruction, Single Data*). La arquitectura MISD describe la organización de un sistema que ejecuta múltiples flujos de instrucciones sobre un flujo de datos único.

Ejemplos de sistemas que implementan este tipo de arquitectura son los sistemas tolerantes a fallos que ejecutan la misma instrucción en paralelo sobre los mismos datos para detectar errores en tiempo de ejecución, como los utilizados en sistemas críticos de aviación.

MIMD (*Multiple Instruction, Multiple Data*). La arquitectura MIMD describe la organización de un sistema que ejecuta múltiples flujos de instrucciones sobre múltiples flujos de datos. Esta arquitectura es la generalización natural de la arquitectura von Neumman.

Existen dos alternativas para implementar un sistema MIMD con varios procesadores y varios dispositivos de memoria:

- Arquitectura MIMD de memoria distribuida o de paso de mensajes. Consiste en replicar los pares procesador/memoria y conectarlos mediante una red de comunicación. Cada memoria sólo puede ser accedida por el procesador que tenga asignado.
- Arquitectura MIMD de memoria compartida. Consiste en construir un conjunto de procesadores y de dispositivos de memoria de forma que cualquier procesador pueda acceder a cualquier dispositivo de memoria a través de un mecanismo de interconexión.

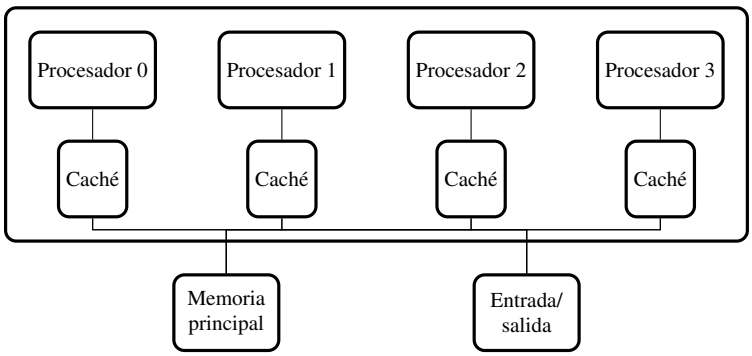

Figura 1: Arquitectura básica de un procesador multinúcleo

6. Paralelismo a nivel de hilo de ejecución

Una de las formas de mejorar el rendimiento de los procesadores es añadir la posibilidad de ejecutar varios hilos de ejecución al mismo tiempo. Un hilo de ejecución es la unidad mínima que puede planificar un sistema operativo. Cada programa tiene al menos un hilo de ejecución; algunos programas pueden tener varios hilos de ejecución.

Este tipo de paralelismo a nivel de hilo es distinto del paralelismo a nivel de instrucción que proporcionan técnicas como el *pipeline* o las arquitecturas superescalares. Dos diferencias fundamentales entre ambos tipos de paralelismo son:

- El paralelismo a nivel de instrucción mejora el rendimiento de cada hilo de ejecución individual, independientemente de otros hilos. El paralelismo a nivel de hilo de ejecución, en cambio, sólo mejora el rendimiento cuando se ejecutan varios hilos a la vez.
- El paralelismo a nivel de instrucción mejora el rendimiento de un programa mono-hilo automáticamente, sin necesidad de cambios en la forma de programar. El paralelismo a nivel de hilo de ejecución requiere transformar los programas mono-hilo en multi-hilo para poder aprovechar las mejoras de rendimiento. La programación multi-hilo es significativamente más compleja que la mono-hilo ya que requiere tener en cuenta la sincronización entre hilos.

Cuando a principios de la primera década del siglo XXI los grandes incrementos en frecuencia dejaron de ser significativos por problemas de disipación de calor, el método más usado para incrementar el rendimiento fue pasar de arquitecturas monoprocesadoras a arquitecturas MIMD de memoria compartida siguiendo el esquema mostrado en la figura 1. Todos los procesadores de este tipo de arquitecturas comparten una misma memoria principal de tal forma que el tiempo de acceso a memoria es similar para todos ellos. Por esta razón, a este tipo de arquitecturas se las suele denominar *Symmetric Shared-Memory Multiprocessors* (SMP) o *Uniform Memory Access* (UMA).

La idea de este tipo de arquitecturas con varios procesadores compartiendo una memoria común es antigua y se solía emplear en servidores, pero la novedad que se produjo a principios del siglo XXI fue que en lugar de tener cada procesador en su propio encapsulado, se introdujeron varios dentro del mismo. Por esta razón, a los procesadores que siguen este diseño se les suele denominar multinúcleo (*multicore*). La tecnología multiprocesador, que antes estaba reservada solo a servidores, es ahora común en todo tipo de computadores de sobremesa y portátiles, y está presente incluso en teléfonos móviles y tabletas.

Tal y como se muestra en la figura 1, habitualmente cada núcleo tiene asociado uno o más niveles de memoria caché. Estas memorias caché son fundamentales para que la memoria principal no se convierta en el cuello de botella de la arquitectura. Pero a su vez, la gestión de estas memorias caché se convierte en uno de los mayores retos, ya que deben estar coordinadas. Por ejemplo, se podría tener un problema si dos procesadores tienen una copia en su caché de una misma posición de memoria principal y uno de ellos la modifica sin que el otro sea informado. Los algoritmos que se encargan de evitar este tipo de problemas se denominan algoritmos de coherencia de caché.

Además de los procesadores multinúcleo, otra forma de explotar el paralelismo a nivel de hilo consiste en replicar ciertas unidades funcionales del procesador, sin llegar a replicar un procesador completo, de tal forma que cuando se cumplan ciertas condiciones sea posible ejecutar instrucciones de varios hilos distintos a la vez en el mismo procesador. A esta tecnología se la denomina *Simultaneous MultiThreading* (SMT) y la implementación más conocida es la tecnología HyperThreading de Intel.

Las dos técnicas de aprovechamiento del paralelismo a nivel de hilo (procesadores multinúcleo y SMT) pueden combinarse, de tal forma que cada núcleo del procesador pueda ejecutar más de un hilo. Por ejemplo, un procesador con cuatro núcleos que implemente la tecnología SMT de tal forma que pueda llegar a planificar dos hilos en cada núcleo ofrece al sistema operativo ocho elementos donde planificar los hilos. Hay que tener en cuenta que estos ocho elementos no son iguales, en el sentido de que no es lo mismo planificar dos hilos dentro del mismo núcleo, donde en ocasiones van a poder ejecutarse a la vez y otras ocasiones (cuando necesiten una misma unidad funcional no replicada) no, que planificarlos en dos núcleos distintos donde no van a competir por las mismas unidades funcionales.