Universidad de Murcia
Departamento de Ingeniería y
Tecnología de Computadores

Apuntes de Ampliación de Estructura de Computadores

Alberto Ros Bardisa

Septiembre de 2024

Resumen

Estos apuntes pretenden servir como guía para la asignatura "Ampliación de Estructura de Computadores" del Grado en Ingeniería Informática y Programación Conjunta de Estudios Oficiales Grado en Matemáticas y Grado en Ingeniería Informática.

Tema 0

Presentación de la asignatura

Esta asignatura pertenece al área de "Arquitectura y Tecnología de Computadores" y es la tercera de una serie de asignaturas obligatorias o troncales del grado de Ingeniería Informática de la Universidad de Murcia destinadas a comprender cómo funcionan los computadores y cómo hacerlos más eficientes.

En la primera de estas asignaturas, "Fundamentos de Computadores", se estudia la codificación de la información y los sistemas digitales combinacionales. Posteriormente, en "Estructura y Tecnología de Computadores" se estudian sistemas digitales secuenciales, el diseño de un procesador básico y la estructura de la jerarquía de memoria. En "Ampliación de Estructura de Computadores" se hace énfasis en el rendimiento del computador y se estudia un procesador más avanzado (segmentado) que el visto en "Estructura y Tecnología de Computadores" y se explican aspectos avanzados de la jerarquía de memoria. Finalmente, en "Arquitectura y Organización de Computadores" se estudia el diseño de un procesador superescalar, la sincronización, el protocolo de coherencia de cachés, los modelos de consistencia de memoria y la red de interconexión.

En particular, en "Ampliación de Estructura de Computadores" vamos a ver cómo se pueden mejorar las prestaciones de los computadores por primera vez en el grado de Ingeniería Informática. Hasta el momento nos hemos centrado en tener un computador que funcione correctamente, sin mirar mucho su rendimiento. En este curso vamos a construir un ordenador mas potente, gracias a mejoras "arquitectónicas". Se puede decir, por tanto, que en esta asignatura comienza lo que se conoce por *arquitectura de computadores*. No solo vamos a aprender a identificar qué diseño es mejor que otro de forma "cualitativa" sino

que también vamos a aprender a decir de forma "cuantitativa" cuantas veces mejor es un diseño que otro, y así podremos evaluar de manera precisa las diversas optimizaciones estudiadas.

La asignatura tiene asignados 6 ECTS. Es decir, se requieren unas 150 horas de trabajo por parte del alumno: 60 de estas horas son presenciales y están repartidas dos horas de teoría y dos horas de práctica (un hora y cuarenta minutos en realidad) a la semana, y 90 de estas horas son no presenciales. Esto último significa que el alumno debe dedicar 90 horas al estudio de la asignatura y realización de ejercicios. En particular, después de cada sesión de dos horas el alumno debe dedicar al rededor de tres horas más al estudio de la asignatura. Haciendo esto semanalmente el alumno llevaría la asignatura al día y sacaría un mayor rendimiento a las horas presenciales.

Tema 1

Análisis de prestaciones en arquitectura de computadores

1.1 Introducción

¿Podríamos medir el rendimiento o prestaciones de un computador con un solo número? En realidad, para ser precisos necesitaríamos más de uno, pero por la necesidad de obtener de forma rápida e intuitiva si un computador ofrece mejores prestaciones que otro, tratamos de definir las prestaciones con un solo número. Por ejemplo, hace pocos años Intel anunciaba una mejora de las prestaciones de sus computadores de última generación de un 19% con respecto a la anterior generación. ¿Qué significa esto en realidad?

En realidad hay muchas métricas importantes a la hora de definir prestaciones. El consumo de energía o el tamaño del procesador son ejemplos de ellas, pero en este tema nos centraremos en el rendimiento. A la hora de medir rendimiento también hay que llevar mucho cuidado y saber las diferentes métricas que nos podemos encontrar, ya que algunas de ellas dependiendo de las circunstancias pueden indicar que un computador con más prestaciones ofrece menos prestaciones.

Pongamos un símil para la realización de una tarea (ejecución de un programa) por un computador. Imaginemos que nuestra tarea es desplazarnos de Murcia a Cartagena. ¿Cómo mediríamos el rendimiento? Podríamos pensar en la velocidad. A mayor velocidad antes acabaremos la tarea. Aunque suene sen-

sato, en realidad esta medida de rendimiento sería una medida parcial y podría dar lugar a conclusiones erróneas, si nos desplazamos más rápido pero por una carretera que hace muchos más kilómetros. Es decir, un coche más rápido que toma una ruta más larga puede tardar más que uno más lento. La medida que nos interesa al final es por qué medio llegamos antes (dejando el consumo de energía –gasolina– a un lado). Es decir, lo que importa es el *tiempo* que tardamos en realizar la tarea. Dejando el símil a un lado, lo que importa es el *tiempo* que tarda el computador en realizar una tarea.

Sin embargo, podríamos pensar también en que no es lo mismo si nos desplazamos en un coche con una capacidad de 5 pasajeros que en un autobús con una capacidad de 50. El autobús, aunque seguramente más lento, muy probablemente realice el trabajo de llevar 100 pasajeros de Murcia a Cartagena antes que el coche. En este caso lo que nos interesa en realidad es el trabajo útil por unidad de tiempo, o lo que es lo mismo, la productividad. De nuevo, dejando el símil a un lado, lo que interesa en caso de tener que realizar varias tareas es la candidad de tareas que se pueden realizar por unidad de tiempo. Fijando el número de tareas, la clave vuelve a estar en el *tiempo*.

1.2 Definición de rendimiento

Podemos definir el rendimiento de forma genérica como la cantidad de trabajo realizado por el computador por unidad de tiempo. Es decir, un ordenador es más rápido que otro cuando realiza la misma cantidad de trabajo en menor tiempo (disminuye el tiempo de respuesta).

$$Rendimiento = \frac{Cantidad\_de\_trabajo}{Tiempo\_de\_ejecución} \tag{1.1}$$

Si estamos interesados en comparar dos computadores que realizan la misma cantidad de trabajo, entonces podemos eliminar de la ecuación 1.1 la cantidad de trabajo y simplemente definir rendimiento como la inversa del tiempo de ejecución.

$$Rendimiento = \frac{1}{Tiempo\_de\_ejecución} \tag{1.2}$$

Ahora, para calcular la mejora de rendimiento de un computador x respecto a otro computador y, ambos procesando la misma carga de trabajo, podemos simplemente dividir el rendimiento de x entre el rendimiento de y, lo que nos

daría cuantas veces es mejor (en términos de rendimiento) x que y. Este factor de mejora se conoce en inglés como *Speedup*. Podremos decir entonces que x es tantas veces más rápido que y.

$$Speedup = \frac{Rendimiento_x}{Rendimiento_y} \tag{1.3}$$

Esta mejora siempre tiene que ser mayor o igual que 1. En caso contrario y sería más rápido que x y tendríamos que alternar el numerador y el denominador para saber cuánto más rápido es y que x.

Sustituyendo el rendimiento por la ecuación 1.2 (o la ecuación 1.1) tenemos también que el *Speedup* se puede obtener dividiendo el tiempo que tarda en ejecutar el trabajo el computador más lento por el tiempo que tarda en ejecutar el trabajo el computador más rápido.

$$Speedup = \frac{Rendimiento_x}{Rendimiento_y} = \frac{\frac{1}{Tiempo\_de\_ejecución_x}}{\frac{1}{Tiempo\_de\_ejecución_y}} = \frac{Tiempo\_de\_ejecución_y}{Tiempo\_de\_ejecución_x} \tag{1.4}$$

A veces, queremos nombrar el Speedup en porcentaje. Para ello simplemente podemos usar la siguente ecuación.

$$Speedup = 1 + \frac{n}{100} \tag{1.5}$$

Así podremos decir que x es un n% más rápido que y.

Ejercicio 1

El computador A tarda 10 segundos en ejecutar un programa mientras que el computador B tarda 15 segundos. Calcula la mejora de prestaciones del computador B sobre el A.

Solución

$$Speedup = \frac{15}{10} = 1.5$$

El computador A es 1.5 veces más rápido que el B.
Expresado en porcentaje:

$$1 + \frac{n}{100} = 1.5$$

$$n = (1.5 - 1) \times 100 = 50$$

El computador A es un 50% más rápido que el B.

1.3 Tiempo de ejecución

El tiempo de ejecución de un programa en segundos se puede calcular de la siguiente forma:

$$T_{ej} = NI \times CPI \times T_c \tag{1.6}$$

donde NI representa el número de instrucciones de un programa, CPI es la media de ciclos que tarda en ejecutarse cada instrucción y T_c es el tiempo de ciclo del computador en segundos.

El tiempo de ciclo (en segundos) no es más que la inversa de la frecuencia del procesador (en hercios), que es quizás una magnitud más común a la hora de describir un computador:

$$T_c(s) = \frac{1}{Freq(Hz)} \tag{1.7}$$

En muchas ocasiones, cuando T_c no varía, es común medir el tiempo de ejecución en ciclos y por tanto su fórmula sería:

$$T_{ej}(ciclos) = NI \times CPI \tag{1.8}$$

El CPI se puede calcular dividiendo los ciclos totales de ejecución entre el número de instrucciones, si sabemos estos datos o mediante una media aritmética ponderada si sabemos cuantos ciclos tarda de media cada tipo de instrucción (CPI_i) y su frecuencia de aparición (f_i):

$$CPI = \frac{ciclos}{NI} = \sum_{i=1}^{n} CPI_i \times f_i \tag{1.9}$$

Ejercicio 2

Tenemos la siguiente distribución de instrucciones:

Tipo	Porcentaje	Ciclos
ALU	33%	2
Load	21%	3
Store	12%	3
Branch	24%	3
FP	10%	12

Sabemos que podemos reducir el número de ciclos de las operaciones de FP a la mitad, pero ello nos provocará un incremento del ciclo de reloj del 25%. Además ahora las cargas consumirán un ciclo más.

a) ¿Cuál de las máquinas es más rápida ahora? ¿En qué factor?.

b) Si conseguimos no afectar a las cargas con la modificación, ¿varía el resultado obtenido? ¿Ahora qué máquina es la más rápida? ¿En qué factor?.

Solución

Se trata de calcular en ambos casos el tiempo de CPU usando la fórmula explicada en la teoría:

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

A partir de los resultados se determina qué máquina es más rápida y se calcula el factor de mejora.

Apartado a

Empezamos por la configuración original (sin aplicar la optimización que describe el enunciado):

$$T_{CPU\_orig} = NI_{orig} \times CPI_{orig} \times T_{c\_orig}$$

donde $CPI_{orig} = 0.33 \times 2 + 0.21 \times 3 + 0.12 \times 3 + 0.24 \times 3 + 0.10 \times 12 = 3.57$, así pues nos queda que:

$$T_{CPU\_orig} = NI_{orig} \times 3.57 \times T_{c\_orig}$$

Para la configuración alternativa el tiempo de ciclo se incrementa un 25% con respecto al original, y los ciclos que tardan las operaciones FP y las cargas se modifican. Tenemos que:

$$T_{c\_alt} = 1.25 \times T_{c\_orig}$$
$$CPI_{alt} = 0.33 \times 2 + 0.21 \times 4 + 0.12 \times 3 + 0.24 \times 3 + 0.10 \times 6 = 3.18$$

El número de instrucciones no cambia ($NI_{alt} = NI_{orig}$), por lo que:

$$T_{CPU\_alt} = NI_{alt} \times 3.18 \times T_{c\_alt} = NI_{orig} \times 3.18 \times 1.25 \times T_{c\_orig} =$$

$$= NI_{orig} \times 3.975 \times T_{c\_orig}$$

Por lo tanto, la configuración original es 1.1134 veces (3.975/3.57) más rápida que la alternativa propuesta, o lo que es lo mismo: un 11.34%.

Apartado b

En este caso hay que volver a calcular el CPI de la configuración alternativa teniendo en cuenta que el número de ciclos de las cargas no se ve afectado:

$$CPI_{alt} = 0.33 \times 2 + 0.21 \times 3 + 0.12 \times 3 + 0.24 \times 3 + 0.10 \times 6 = 2.97$$
$$T_{CPU\_alt} = NI_{orig} \times 2.97 \times 1.25 \times T_{c\_orig} = NI_{orig} \times 3.7125 \times T_{c\_orig}$$

La configuración original sigue siendo más rápida, esta vez en un factor 1.04 (3.7125/3.57).

1.4 Métricas de rendimiento populares

El tiempo de ejecución es la métrica más completa a la hora de medir el rendimiento, pero también es común el uso de otras métricas como el CPI (o su inversa el IPC – instrucciones por ciclo), los MIPS o los MFLOPS. Estas métricas son útiles a la hora de medir prestaciones pero hay que llevar cuidado con ellas en algunos casos, ya que pueden no ser representativas de la mejora del computador.

Los MIPS o millones de instrucciones por segundo es una métrica de veloci-

dad es fáciles de comprender ya que más MIPS significa más rápido. Podemos describir esta métrica de las dos formas siguientes:

$$MIPS = \frac{NI \times 10^{-6}}{T_{ej}} = \frac{Frec(MHz)}{CPI} \tag{1.10}$$

Dejamos al lector el razonamiento de por qué ambas formulas son equivalentes, una vez hemos explicado la fórmula del tiempo de ejecución en el apartado anterior.

Los MIPS sin embargo tienen una serie de desventajas importantes: (1) son dependientes del conjunto de instrucciones, (2) varían entre programas en el mismo ordenador y (3) pueden variar inversamente al rendimiento (por ejemplo, un computador puede ofrecer más MIPS a costa ejecutar más instrucciones pero más sencillas para realizar la misma tarea que otro programa con menos instrucciones más lentas). Por ejemplo, sería injusto comparar un computador que implementa una arquitectura RISC con otro que implementa una arquitectura CISC usando MIPS.

Volviendo al símil del coche, los kilómetros recorridos por el coche sería el equivalente al número de instrucciones ejecutadas por el programa, mientras que el tiempo sigue siendo la métrica relevante para medir el rendimiento. Los MIPS siendo una métrica de velocidad, sería equivalente a la velocidad del coche (e.g., Km/h). Sin embargo esta métrica no es representativa del tiempo si la distancia que recorren dos coches para llegar al mismo sitio (número de instrucciones ejecutadas por el programa) no es la misma.

Los MIPS ganan relevancia para comparar dos computadores que ejecutan en mismo binario, es decir, el mismo número de instrucciones NI, ya que si miramos la última fórmula de la ecuación 1.10 podemos ver que se parece a la inversa del tiempo de ejecución pero sin tener en cuenta el NI.

Ejercicio 3

Al compilar un programa con un compilador estándar y ejecutarlo en nuestra máquina, el programa ejecuta un 43% de instrucciones de ALU (que tardan 1 ciclo de media), 21% de cargas (2 ciclos), 12% de almacenamientos (2 ciclos) y 24% de saltos condicionales (2 ciclos). Hemos conseguido un compilador más avanzado que es capaz de descartar el 50% de las instrucciones de la ALU. Suponiendo que nuestra máquina tiene una duración del ciclo de reloj $T_c =$

20 ns, es decir 50 MHz, ¿cuál es el rendimiento en MIPS del código optimizado frente al código original? Razona si el código con más MIPS es el que menor tiempo de ejecución obtendría.

Solución

Los MIPS se pueden calcular como:

$$MIPS = \frac{Freq(MHz)}{CPI}$$

El CPI con el compilador estándar sería:

$$CPI_{std} = 0.43 \times 1 + 0.21 \times 2 + 0.12 \times 2 + 0.24 \times 2 = 1.57$$

Con lo que los MIPS serían:

$$MIPS_{std} = \frac{50}{1.57} = 31.85$$

El CPI con el compilador optimizado sería:

$$CPI_{opt} = \frac{0.215 \times 1 + 0.21 \times 2 + 0.12 \times 2 + 0.24 \times 2}{0.215 + 0.21 + 0.12 + 0.24} = 1.73$$

Con lo que los MIPS serían:

$$MIPS_{opt} = \frac{50}{1.73} = 28.96$$

El programa optimizado sería más lento según los MIPS, pero claramente podemos pensar que su tiempo de ejecución sería menor, ya que ejecuta menos instrucciones que el programa compilado con el compilador estándar.

El CPI o IPC, es una métrica más limitada que los MIPS, pero es representativa del tiempo de ejecución cuando no varía ni el NI ni el T_c, es decir cuando queremos medir las prestaciones de cambios en el diseño que no afectan a la frecuencia del procesador.

Los MFLOPS o millones de operaciones de punto flotante por segundo es otra métrica de velocidad que se considera en algunos casos más relevante que los MIPS, ya que su principal ventaja es que el número de instrucciones de punto

flotante ($NIP.F.$) depende del algoritmo y no de la máquina. Su fórmula es la siguiente:

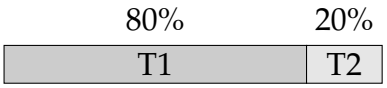
$$MFLOPS = \frac{NI\ P.F. \times 10^{-6}}{T_{ej}} \tag{1.11}$$

Los MFLOPS es también una métrica popular y se usa por ejemplo en el `top500.org` para medir las prestaciones de los supercomputadores más potentes del planeta. Sin embargo, esta métrica también tiene una serie de desventajas: (1) no se pueden aplicar a todos los programas (por ejemplo un compilador no usa operaciones de punto flotante), (2) todas las máquinas no tienen las mismas operaciones de punto flotante y (3) todas las operaciones de punto flotante no tardan lo mismo (por ejemplo una división tarda mucho más ciclos que una suma).

1.5 La Ley de Amdahl

La Ley de Amdahl no es más que una fórmula para medir el Speedup de un computador respecto a otro, cuando sabemos la fracción de tiempo que se mejora y en cuanto se mejora.

Imaginemos una aplicación en la que el 80% del tiempo transcurre realizando una tarea (T1) y el 20% restante transcurre realizando otra tarea (T2).



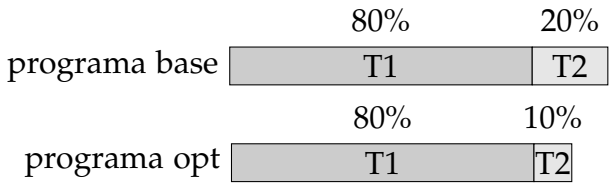
Imaginemos también que, con mucho esfuerzo, hemos conseguido eliminar completamente la necesitad de ejecutar la tarea T2. ¿Cuál será el Speedup o mejora del rendimiento de la aplicación?

Si la aplicación antes (llamémosla base) tardaba x segundos y la aplicación ahora (llamémosla optimizada u opt) tarda $0.8x$ segundos (un 80% del tiempo de la aplicación base), entonces tenemos que:

$$Speedup = \frac{Tiempo_{base}}{Tiempo_{opt}} = \frac{x}{0.8x} = \frac{1}{0.8} = 1.25 \ \ (25\%)$$

Consideremos ahora que en vez de eliminar completamente la tarea T2 hemos conseguido hacerla tan solo el doble de rápido. ¿Cuál será el *speedup* obtenido en este caso con respecto a la aplicación base?

Si la tarea se ejecuta el doble de rápido, tardará la mitad del tiempo, y por tanto tenemos:

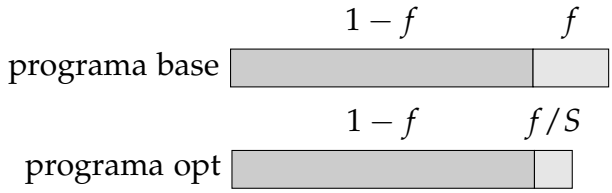


donde el programa optimizado muestra porcentajes relativos al tiempo del primer programa. El *speedup* en este caso sería:

$$Speedup = \frac{Tiempo_{base}}{Tiempo_{opt}} = \frac{x}{0.9x} = \frac{1}{0.9} = 1.11$$

Para calcular este *speedup* hemos hecho uso de la Ley de Amdahl.

La Ley de Amdahl es una fórmula matemática que describe la mejora total en el rendimiento (o *speedup*) de un sistema (computador en nuestro caso) con respecto a la mejora de una fracción de tiempo de dicho sistema, es decir, si mejoramos una tarea cuya fracción de tiempo (f) se conoce en un factor S, la siguiente fórmula nos permite saber el porcentaje de mejora del sistema.



$$Speedup = \frac{Tiempo_{base}}{Tiempo_{opt}} = \frac{(1-f)+f}{(1-f)+\frac{f}{S}} = \frac{1}{(1-f)+\frac{f}{S}} \tag{1.12}$$

Por ejemplo, si en un procesador conseguimos mejorar las instrucciones en coma flotante para que se ejecuten el doble de rápidas, pero el procesador ejecuta una aplicación que solo dedica un 10% de su tiempo a ejecutar estas instrucciones podemos aplicar la Ley de Amdahl, ya que sabemos el porcentaje de tiempo de la parte mejorada (10% del total) y sabemos en qué factor estamos mejorando (el doble, o sea un factor de 2).

$$Speedup = \frac{1}{(1-f)+\frac{f}{S}} = \frac{1}{(1-0.1)+\frac{0.1}{2}} = 1.053 \;\; (5.3\%)$$

Cabe destacar que la mejora podría cambiar al ejecutar otra aplicación ya que el porcentaje del tiempo que dedica a las instrucciones de coma flotante podría variar.

Ejercicio 4

Supongamos que se puede mejorar la velocidad de la CPU de nuestra máquina en un factor de cinco (sin afectar al rendimiento de la E/S) por cinco veces el coste. Supongamos también que la CPU se utiliza el 50% del tiempo, y que el tiempo restante la CPU está esperando las operaciones de E/S. ¿Cuánto se incrementa el rendimiento total? Si la CPU supone un tercio del coste total de la máquina, ¿cuánto se incrementa el coste total? Por tanto, ¿el incremento de la velocidad de la CPU en un factor de cinco es una buena inversión desde un punto de vista coste/rendimiento?

Solución

Inicialmente, la relación coste/rendimiento de la máquina es C/R. Vamos a calcular cómo se incrementa el rendimiento por un lado y el coste por otro con el cambio. Si el rendimiento se incrementa en mayor medida que el coste, o ambos aspectos se ven mejorados en igual proporción, entonces el cambio es interesante desde el punto de vista de la relación coste/rendimiento. Si el coste se incrementa en mayor medida que el rendimiento, entonces el cambio no es aconsejable.

Para calcular la mejora en el rendimiento que se obtiene con el cambio aplicamos la Ley de Amdahl, considerando que la CPU se usa el 50% del tiempo:

$$Speedup = \frac{1}{1 - 0.5 + 0.5/5} = 1.667$$

Lo que significa que el rendimiento tras el cambio $R' = 1.667 \times R$.

Por otro lado, la CPU supone $1/3$ del coste del sistema y se ha multiplicado por 5 su precio, con lo que el coste tras el cambio $C' = 2/3 \times C + 1/3 \times C \times 5 = 7/3 \times C = 2.334 \times C$

Concluimos, pues, que el cambio no es interesante, ya que el coste se incrementa en mayor proporción que el rendimiento (incremento de 2.334 veces en el coste para obtener un factor de mejora en el rendimiento de 1.667).

Como conclusión, la Ley de Amdahl sirve para guiar al arquitecto de computadores sobre qué partes del computador optimizar (aquellas a las que dedique más tiempo).

1.6 Cómo obtener el rendimiento medio

En esta sección veremos como obtener el rendimiento medio cuando ejecutamos varias aplicaciones que ofrecen diferentes prestaciones dependiendo de la máquina donde se ejecuten. Esto es muy interesante ya que siempre es conveniente mostrar la mejora de prestaciones usando un sólo número, aunque no es siempre una tarea sencilla.

Imaginemos, por ejemplo, que tenemos dos computadores (A y B) y dos aplicaciones (1 y 2) que usamos para evaluar el rendimiento de los computadores. Al ejecutar las aplicaciones en ambos computadores obtenemos los siguientes tiempos:

	Computador A	Computador B
Aplicación 1	10 segundos	2 segundos
Aplicación 2	20 segundos	60 segundos

¿Qué computador obtiene mejor rendimiento? ¿Cuál preferiríais comprar? Como usuario podríais pensar que vais a ejecutar prácticamente solo la aplicación 1 y por tanto comprar el computador B. Pero para el fabricante que no sabe qué aplicación va a usar más el usuario la decisión de si disenar un procesador como el A o uno como el B puede no ser tan sencilla. Para tomar esta decisión se podría obtener la media del rendimiento de las aplicaciones para cada computador, pero medias hay muchas y hay que saber usar la media más adecuada en cada caso.

En esta sección comentaremos diferentes medias (aritmética, aritmética ponderada, armónica, armónica ponderada y geométrica), así como sus ventajas e inconvenientes.

1.6.1 Media aritmética

La media aritmética equivale a usar para obtener el rendimiento la suma de los tiempos de ejecución de todas las aplicaciones.

Para obtener la media aritmética usaríamos la siguiente fórmula:

$$Media\_aritmética = \sum_{i=1}^{n} \frac{tiempo_i}{n} \tag{1.13}$$

Esta media es adecuada cuando disponemos de los tiempos de cada programa pero no sabemos con qué frecuencia se va a ejecutar cada uno.

En el ejemplo anterior, la media aritmética para el computador A y B sería:

	Computador A	Computador B
Aplicación 1	10 segundos	2 segundos
Aplicación 2	20 segundos	60 segundos
Media aritmética	15 segundos	31 segundos

Por tanto, podríamos decir que el computador A es $\frac{31}{15} = 2.07$ veces (o un 107%) más rápido que el computador B.

1.6.2 Media aritmética ponderada

No todos los programas se ejecutan el mismo número de veces en una máquina. Si supiéramos la frecuencia de ejecución de cada programa, podríamos obtener una mejor medida del rendimiento del computador aplicando media aritmética ponderada, cuya fórmula se muestra a continuación:

$$Media\_aritmética\_ponderada = \sum_{i=1}^{n} w_i \times tiempo_i \tag{1.14}$$

donde w_i es el ratio o frecuencia de ejecución de cada aplicación (entre 0 y 1).

Imaginemos ahora que sabemos que el usuario final va a ejecutar la aplicación 1 un 10% del tiempo mientras que la aplicación 2 el 90% del tiempo restante.

En ese caso podríamos calcular la media aritmética ponderada de la siguiente manera:

	Computador A	Computador B
Aplicación 1 (w = 0.1)	10 segundos	2 segundos
Aplicación 2 (w = 0.9)	20 segundos	60 segundos
	$0.1 \times 10 + 0.9 \times 20$	$0.1 \times 2 + 0.9 \times 60$
Media aritmética ponderada	19 segundos	54.2 segundos

Podríamos decir en este caso que el computador A es $\frac{54.2}{19} = 2.85$ veces (o un 185%) más rápido que el computador B.

Si por el contrario la aplicación 1 se ejecutara un 90% de las veces y la aplicación 2 el 10% restante, la media aritmética ponderada quedaría como:

	Computador A	Computador B
Aplicación 1 (w = 0.9)	10 segundos	2 segundos
Aplicación 2 (w = 0.1)	20 segundos	60 segundos
	$0.9 \times 10 + 0.1 \times 20$	$0.9 \times 2 + 0.1 \times 60$
Media aritmética ponderada	11 segundos	7.8 segundos

En este caso deberíamos decir que el computador B es $\frac{11}{7.8} = 1.41$ veces (o un 41%) más rápido que el computador A.

Como vemos el rendimiento puede cambiar dependiendo de la frecuencia de ejecución de los programas.

1.6.3 Media armónica

Cuando en vez de tiempos de ejecución disponemos de velocidades o frecuencias, como es el caso de métricas como MIPS, MFLOPS, o IPC, no debemos usar la media aritmética, ya que no es la adecuada para este tipo de magnitudes.

Pongamos de nuevo el ejemplo del coche que tiene que hacer un recorrido esta vez de ida y vuelta. Si el coche viaja a la ida a 50km/h y a la vuetla a 100km/h, ¿cuál ha sido su velocidad media? Definitivamente no es 75km/h, valor que se obtendría al calcular la media usando la media aritmética. La razón es que el tiempo que tarda en ir y en volver no es el mismo. El coche ha estado más tiempo yendo a una velocidad de 50km/h que de 100km/h. La media por tanto debería estar más cerca de 50km/h que de 100km/h.

Para calcular la media en este caso debemos aplicar la media armónica, cuya fórmula es la siguiente:

$$Media\_armónica = \frac{n}{\sum_{i=1}^{n} \frac{1}{velocidad_i}} \tag{1.15}$$

Y el resultado sería la velocidad media de velocidad del coche:

$$Media = \frac{2}{\frac{1}{50} + \frac{1}{100}} = 66.67\ km/h$$

Por tanto, la media armónica es la adecuada en caso de disponer de MIPS, MFLOPS o IPC, ya que son métricas de velocidad.

1.6.4 Media armónica ponderada

Si tuviéramos información sobre las veces que se va a ejecutar cada programa y la velocidad de cada uno en cada computador, entonces al igual que pasaba con la media aritmética, podemos ponderarla, y por tanto usar la media armónica ponderada, cuya fórmula es la siguiente:

$$Media\_armónica\_ponderada = \frac{n}{\sum_{i=1}^{n} \frac{w_i}{velocidad_i}} \tag{1.16}$$

1.6.5 Media geométrica

A veces es interesante normalizar los resultados obtenidos, con el fin de que una aplicación que tarda mucho más que otra no tenga necesariamente más peso en la media final. Normalizar significa dividir los resultados de las aplicaciones por los obtenidos por dichas aplicaciones en una máquina de referencia. Por otro lado, el Speedup es una métrica muy usada para ver la mejora de un computador respecto a otro. Tanto la normalización como el Speedup son equivalentes, ya que representan la fracción de mejora pero no tienen unidades. En este caso ¿qué media deberíamos usar? La verdad es que la comunidad científica no se pone de acuerdo al respecto, pero sí que hay una cosa clara: la media geométrica tiene la ventaja en este caso de que independientemente de la máquina de normalización el Speedup va a ser coherente, es decir, no cambia. Esto sin embargo no pasa con las medias aritméticas o armónicas, las cuales dependiendo de la máquina de referencia (para la que normalizamos) el resultado puede ser que una máquina es mejor o que la otra es mejor.

La media geométrica se calcula como la raíz enésima del producto de los n valores disponibles:

$$Media\_geométrica = \sqrt[n]{\prod_{i=1}^{n} tiempo\_normalizado_i} \tag{1.17}$$

Veamos la coherencia de la media geométrica mediante un ejemplo. Normalicemos los resultados de tiempo anteriores respecto al computador A, dividiendo todos los tiempos por aquellos obtenidos por el computador A:

	Computador A	Computador B	Norm $A(/A)$	Norm $B(/A)$
Aplicación 1	10 segundos	2 segundos	1	0.2
Aplicación 2	20 segundos	60 segundos	1	3

Ahora calculemos la media de los valores normalizados respecto al computador A usando tanto la media aritmética como la geométrica:

	Computador A	Computador B	Norm $A(/A)$	Norm $B(/A)$
Aplicación 1	10 segundos	2 segundos	1	0.2
Aplicación 2	20 segundos	60 segundos	1	3
Media aritmética			1	1.6
Media geométrica			1	0.77

Podemos ver que la media aritmética me indica que el computador A es $\frac{1.6}{1} = 1.6$ veces (o un 60%) más rápido que B. Sin embargo, la media geométrica me indica que el computador B es $\frac{1}{0.77} = 1.29$ veces (o un 29%) más rápido que A. Parece ser que las medias no se ponen de acuerdo, pero es complicado decir cual es la correcta.

Veamos ahora qué sucede si normalizamos los tiempos con respecto al computador B:

	Computador A	Computador B	Norm $A(/B)$	Norm $B(/B)$
Aplicación 1	10 segundos	2 segundos	5	1
Aplicación 2	20 segundos	60 segundos	0.33	1
Media aritmética			2.67	1
Media geométrica			1.29	1

En este caso, mientras que la media geométrica me sigue diciendo que el computador B es $\frac{1.29}{1} = 1.29$ veces más rápido que el computador A, ahora la media aritmética cambia de versión e indica que que el computador B es $\frac{2.67}{1} = 2.67$ veces (o un 167%) más rápido que el computador A.

Como vemos la media geométrica es independiente de la máquina respecto a la que normalizamos, pero la media aritmética no.

Por otro lado, es importante destacar que la media geométrica no es representativa de los tiempos de ejecución (la aritmética sí), aunque ofrezca resultados coherentes.

1.7 Programas de prueba (benchmarks)

Los benchmarks son programas de prueba que sirven como instrumento para medir el rendimiento de un sistema o de sus componentes.

Estos programas de prueba pueden ser diseñados específicamente para medir cualquiera de los componentes del sistema, ya sea la CPU, tarjeta grafica, RAM, etc.

Existen distintos tipos de benchmarks con sus ventajas e inconvenientes:

- Aplicaciones reales: aunque son las más precisas a la hora de obtener el rendimiento, pueden tener problemas de portabilidad.
- Aplicaciones modificadas: solucionan los problemas de portabilidad de las aplicaciones reales y se centran en un aspecto concreto del sistema.
- Núcleos (kernels): son pequeñas partes de cruciales de programas reales, que se ejecutan frecuentemente.
- Benchmarks reducidos (toys): Son benchmarks pequeños y fácilmente portables, aunque poco útiles para evaluar el rendimiento.
- Benchmarks sintéticos: si objetivo es obtener un perfil medio de ejecución, por ejemplo se puede indicar cuantas operaciones de lectura el programa va a tener. No son adecuados para obtener medidas de rendimiento fiables.

Existen algunos conjuntos de aplicaciones (o benchmark suites) usados ampliamente para evaluar prestaciones:

- SPEC (Standard Performance Evaluation Coorporation): Tiene como objetivo crear un conjunto de benchmarks estándar para medir el rendimiento de computadores y controlar los resultados de los tests realizados, que posteriormente se hacen públicos. Principalmente enfocado en medir el rendimiento de la CPU.
- TPC (Transaction Processing Council): Modelan usuarios que ejecutan transacciones en una base de datos. El TPC mide la rapidez y fiabilidad con la que se realizan dichas transacciones. La unidad de medida son transacciones por minuto o segundo.

Tema 2

Segmentación básica

2.1 Introducción

Concepto de segmentación. Segmentación para la ejecución de instrucciones: camino de datos y control segmentado. Riesgos de la segmentación (estructurales, de datos y de control). Excepciones y su implementación.

2.1.1 El juego de instrucciones DLX

Repaso de instrucciones. Remarcar diferencias con MIPS (ETC): blt -> slt & beqz

2.1.2 Procesador multiciclo para DLX

Repaso de etapas multiciclo (ETC), pero adaptado a DLX. Ejemplo de un load, que es el más largo (5 ciclos). El resto de instrucciones se adaptan a éste.

2.2 Segmentación para la ejecución de instrucciones

Describir segmentación a grandes rasgos, o de forma esquemática. Los detalles de implementación se verán al final.

Ejercicio cuantitativo de segmentación.

2.3 Problemas de la segmentación: los riesgos

Si tuvieramos una fábrica de coches la segmentación ya funcionaría bien y tendríamos un cauce ideal, produciendo un coche por tiempo de cada etapa.

Pero las instrucciones son más complicadas. No todas son iguales. Y la ejecución de una puede depender de la otra.

Hay 3 tipos de riesgos: estructurales, de datos y de control.

2.3.1 Riesgos estructurales

Opciones: duplicar, parar, evitar sw (separar o nops).

2.3.2 Riesgos de datos

Opciones: parar, forwarding, evitar sw (separar o nops).

2.3.3 Riesgos de control

Opciones: parar, adelantar calculo salto, predecir, evitar sw (delay-slot o nops).

Detalle de la implementación. ¿Cómo parar ante riesgos (y adelantar el calculo del salto)?

Ejercicios de delay en calculo de tiempo.

Tema 3

Segmentación avanzada y predicción de saltos

3.1 Introducción

Mitigación de los riesgos de datos: adelantamientos. Mitigación de los riesgos de control (predicción de saltos estática y predicción de saltos dinámica). Segmentación DLX de punto flotante. Cauces con terminación fuera de orden.

3.2 Predicción de saltos

Observación 1: los saltos tienden a tener el mismo comportamiento.

Observación 2: saltos correlacionados.

3.2.1 Estática

Algo más sobre predicción estática saltos para alante o para atrás o comparación == 0 o != 0.

3.2.2 Contador saturado por salto

Ejemplo: bucle.

Aliasing.

3.2.3 Correlación

ejemplo: if d == 0 ejemplo: found = find(elem); if (found) ... if a[i] = elem.
historia global.
gshare.

Tema 4

Sistema de memoria de altas prestaciones

4.1 Introducción

Introducción a los sistemas de memoria. Diseño de una jerarquía de caches de alto rendimiento: reducción de la tasa de fallos, reducción de la penalización por fallo y reducción del tiempo de acierto en caches. Organizaciones de la memoria principal. Memoria virtual. Técnicas de traducción rápida de direcciones virtuales. Protección de memoria.

4.2 Evaluación del Rendimiento de la Jerarquía de Memoria

4.3 Reducción de la Tasa de Fallos de Caché

4.3.1 Clasificación de fallos de caché

Podemos distinguir los siguientes motivos por los que se producen fallos en caché:

Forzosos: el primer acceso a un bloque de memoria no puede estar en la

caché (poco efecto en programas grandes) - Llamados fallos de arranque en frío o de primera referencia

Capacidad: la memoria caché no tiene el tamaño suficiente para contener todos los bloques necesarios - En un momento determinado uno de los bloques que se han utilizado tiene que dejar hueco a otro (la caché está llena) - Se produce fallo si se vuelve a referenciar el bloque desalojado

Conflicto: la memoria caché no es totalmente asociativa - Aciertos en una caché totalmente asociativa que se vuelven fallos en una asociativa por conjuntos de n-vías se deben a más de n peticiones sobre algunos conjuntos

4.3.2 Optimizaciones hardware

4.3.2.1 Aumento del tamaño de bloque

Y del tamaño de la caché. De hecho la primera figura (pag 29) lo contempla.

4.3.2.2 Aumento de la asociatividad

4.3.2.3 Caché de víctimas

4.3.2.4 Búsqueda anticipada de datos e instrucciones (prefetching)

4.3.3 Optimizaciones del compilador

4.3.3.1 Combinación de arrays (merging arrays)

4.3.3.2 Intercambio de iteraciones (loop exchange)

4.3.3.3 Unión de bucles (loop fusion)

4.3.3.4 Blocking

4.3.3.5 Búsqueda anticipada

4.4 Reducción de la Penalización por Fallo de Caché

4.5 Reducción del Tiempo en Caso de Acierto en Caché

4.6 Organizaciones de la Memoria Principal

Bibliography