

terminar una instrucción por ciclo de reloj como máximo, su velocidad pico sería $(1 \text{ instrucción/ciclo})/1 \times 10^{-9} \text{ (segundos/ciclo)} = 1 \times 10^9 \text{ instrucciones/segundo}$, o lo que es lo mismo 1 GIPS. Es decir, $(1 \times 10^9 \text{ instrucciones})/(1 \text{ segundo} \times 10^9)$.

En resumen, a no ser que los MIPS (o los GIPS) se utilicen como medida de velocidad pico para comparar procesadores con el mismo repertorio de instrucciones, la única forma de usar esta medida para comparar dos máquinas es que, además de que ambas tengan el mismo repertorio, se use el mismo programa (con las mismas instrucciones máquina generadas por el compilador).

En relación a la pregunta que se plantea, si un programa que ejecuta 32.000 instrucciones tarda 1 microsegundo, tendríamos que, efectivamente, el programa se ha ejecutado a una velocidad de 32 GIPS:

$$GIPS = \frac{32.000 \text{ instrucciones}}{1 \times 10^{-6} \text{ segundos} \times 10^9} = \frac{32 \times 10^3}{1 \times 10^3} = 32$$

No obstante, NO podríamos estar seguros de que cualquier programa de 32000 instrucciones tardaría en ejecutarse un microsegundo porque dependería de las características de las instrucciones que lo compongan.

Sin ir más lejos, el número de instrucciones que constituyen un programa (número estático de instrucciones) puede ser distinto del número de instrucciones que finalmente ejecuta el procesador (número dinámico de instrucciones), ya que puede haber instrucciones de salto, bucles, etc. que hacen que ciertas instrucciones del código se ejecuten más de una vez, y otras no se ejecuten nunca.

Por otro lado, según el tipo de instrucciones que constituyan el programa y las dependencias entre dichas instrucciones, pueden variar los tiempos que tardan en ejecutarse las instrucciones.

Problema 1. En el bucle siguiente, los arrays $a[]$, $b[]$, $c[]$, y $d[]$, son números en coma flotante de 64 bits, y $n=2 \times 10^{10}$:

```
for (i = 0 ; i < n ; i++)
    d[i] = a[i] + b[i] + c[i];
```

Si el programa se ejecuta en un procesador a 2 GHz que puede terminar dos operaciones en coma flotante por ciclo, ¿cuál es el tiempo mínimo que tardaría en ejecutarse?. ¿Cuántos GFLOPS de velocidad pico tiene el procesador?.

Solución

El tiempo mínimo que tardaría en ejecutarse el programa se puede estimar teniendo en cuenta el número máximo de instrucciones en coma flotante por ciclo que puede terminar el programa. Obviamente, teniendo en cuenta que el programa debe ejecutar

más instrucciones (las de carga de memoria, control del bucle, etc.) y que hay que realizar accesos a memoria que, en ciertos casos darán lugar a faltas de caché y consumirán su tiempo correspondiente, la ejecución del programa consumirá más tiempo que el que obtengamos considerando sólo la ejecución de operaciones en coma flotante, pero lo que pretendemos es obtener una estimación del tiempo mínimo de ejecución.

$$T_{CPU}(\text{min}) \approx N_{\text{float}} \times CPI_{\text{float}} \times T_{\text{ciclo}} \approx \frac{N_{\text{float}}}{IPC_{\text{float}} \times F}$$

donde N_{float} es el número de operaciones en coma flotante, CPI_{float} es el número de ciclos por operación en coma flotante, que es igual a la inversa del número de operaciones o instrucciones en coma flotante por ciclo, IPC_{float} , y T_{ciclo} es tiempo por ciclo, es decir, la inversa de la frecuencia de reloj, F .

Por tanto, dado que $N_{\text{float}} = 2 \times 10^{10}$, que $IPC_{\text{float}} = 2$ (terminan dos operaciones en coma flotante por ciclo), y que $F = 2 \times 10^9$ ciclos/segundo obtenemos:

$$T_{CPU}(\text{min}) \approx \frac{2 \times 10^{10} \text{ op. com. flot.}}{2 \left(\frac{\text{op. com. flot.}}{\text{ciclo}} \right) \times \left(2 \times 10^9 \frac{\text{ciclos}}{\text{segundo}} \right)} = 5 \text{ segundos}$$

En cuanto al número de GFLOPS pico del procesador, dado que puede terminar como máximo dos operaciones en coma flotante por ciclo, es decir $IPC_{\text{float}} = 2$ operaciones en coma flotante por ciclo, tendremos que:

$$\begin{aligned} GFLOPS_{\text{pico}} &= IPC_{\text{float}} \left(\frac{\text{op. com. flot.}}{\text{ciclo}} \right) \times F \left(\frac{\text{ciclos}}{\text{segundo}} \right) \times 10^{-9} = \\ &= 2 \left(\frac{\text{op. com. flot.}}{\text{ciclo}} \right) \times \left(2 \times 10^9 \frac{\text{ciclos}}{\text{segundo}} \right) \times 10^{-9} = 4 \end{aligned}$$

Por tanto, el procesador tiene 4 GFLOPS de velocidad pico (se terminan 2 operaciones en coma flotante por ciclo y cada ciclo es de 0.5 ns).



Problema 2. En un procesador sin segmentación de cauce, se plantean dos alternativas en el repertorio de instrucciones máquina para implementar los saltos condicionales:

- ALT1. Utilizar dos instrucciones: una instrucción de comparación COMP actualiza un código de condición y es seguida por una instrucción de salto BRANCH que comprueba esa condición.
- ALT2. Utilizar una sola instrucción: esa instrucción (COMP+BRANCH) incluiría la funcionalidad de las instrucciones COMP y BRANCH de la alternativa ALT1.

Indique qué alternativa es la mejor teniendo en cuenta que hay un 30% de instrucciones BRANCH en los programas con la primera alternativa (ALT1); que las instrucciones BRANCH (de ALT1) y COMP+BRANCH (de ALT2) necesitan 4 ciclos mientras que todas

las demás necesitan sólo 3; y que el ciclo de reloj de la ALT1 es un 15% menor que el de la ALT2, dado que al incluirse en la alternativa ALT2 una instrucción de mayor funcionalidad (la instrucción COMP+BRANCH) la microarquitectura del procesador es algo más compleja en este caso y el hardware correspondiente algo más lento (los diseños de las microarquitecturas utilizan el mismo número de ciclos para las instrucciones del mismo tipo en las dos alternativas, pero el ciclo de reloj es mayor en la microarquitectura del procesador de la ALT2).

Solución

Para resolver este problema determinaremos el tiempo medio de CPU para cada una de las alternativas y los compararemos. La alternativa para la que ese tiempo medio de CPU sea menor será la mejor. El tiempo medio de CPU se puede expresar como:

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

donde CPI es el número medio de ciclos por instrucción, NI es el número de instrucciones, y T_{CICLO} el tiempo de ciclo del procesador. La situación en ALT1 se resume en la Tabla 1

Tabla 1. Situación de la ALT1

Tipo de Instrucción	Número de Instrucciones	Ciclos de cada tipo de instrucción
BRANCH	$0.3NI_1$	4
Resto	$0.7NI_1$	3
	$NI_1 = 0.3NI_1 + 0.7NI_1$	$CPI_1 = 0.3 \times 4 + 0.7 \times 3 = 3.3$

y tendríamos que calcular:

$$T_{CPU}(1) = CPI_1 \times NI_1 \times T_{CICLO}(1)$$

El valor de CPI_1 se obtiene sumando, para todos los tipos de instrucciones, los productos del número de ciclos que necesita cada instrucción en ALT1 y la frecuencia con que aparece dicha instrucción:

$$CPI_1 = \left(\frac{0.3 \times NI_1}{NI_1} \times 4 \right) + \left(\frac{0.7 \times NI_1}{NI_1} \times 3 \right) = 3.3$$

y, por tanto

$$T_{CPU}(1) = 3.3 \times NI_1 \times T_{CICLO}(1)$$

A continuación se calcula el tiempo medio de CPU para la segunda alternativa, ALT2:

$$T_{CPU}(2) = CPI_2 \times NI_2 \times T_{CICLO}(2)$$

En realidad, lo que se va a hacer es expresarlo en términos del número medio de ciclos por instrucción, el número de instrucciones, y el tiempo de ciclo de la alternativa ALT1 para poder comparar ambas expresiones y determinar cuál es la mejor.



Así, para determinar el número de instrucciones que habrá en la ALT2 hay que tener en cuenta lo siguiente:

- En los códigos de ALT2 tendremos $0.3NI_1$ instrucciones COMP+BRANCH dado que ALT1 se hacía utilizando sus instrucciones BRANCH.
- En los códigos de ALT2 no se tendrán las instrucciones COMP, que en los códigos de ALT1 estaban asociadas a instrucciones BRANCH para implementar el salto condicional, ya que esta función se encuentra incluida en las instrucciones COMP+BRANCH que se utiliza en ALT2. Esto quiere decir que habrá $0.4NI_1$ instrucciones distintas de las instrucciones COMP+BRANCH en ALT2, es decir $0.7NI_1 - 0.3NI_1$ (restamos a las instrucciones que había en ALT1 las COMP que desaparecen en ALT2).

Así, el número de instrucciones que quedan en ALT2 son

$$NI_2 = 0.3NI_1 + (0.7NI_1 - 0.3NI_1) = 0.7NI_1$$

y, dado que el número de ciclos de las instrucciones BRANCH de ALT1 y las COMP+BRANCH de ALT2 es igual a 4 y que el resto de instrucciones mantienen el número de ciclos que necesitan, el valor de CPI_2 es:

$$CPI_2 = \left(\frac{0.3 \times NI_1}{0.7 \times NI_1} \times 4 \right) + \left(\frac{0.4 \times NI_1}{0.7 \times NI_1} \times 3 \right) = \left(\frac{0.3}{0.7} \times 4 \right) + \left(\frac{0.4}{0.7} \times 3 \right) = 3.43$$

La situación de la ALT2 se resume en la Tabla 2

Tabla 2. Situación de la ALT2

Tipo de Instrucción	Número de Instrucciones	Ciclos de cada tipo de instrucción
COMP+BRANCH	$0.3NI_1$	4
Resto	$(0.7NI_1 - 0.3NI_1) = 0.4NI_1$	3
	$NI_2 = 0.3NI_1 + 0.4NI_1 = 0.7NI_1$	$CPI_2 = 0.43 \times 4 + 0.57 \times 3 = 3.43$

Además, se tiene que el ciclo de reloj de ALT2 es más largo que el de ALT1. Concretamente, se dice en el enunciado que el ciclo de ALT1 es un 15% menor que el de ALT2, lo que implica que:

$$T_{CICLO}(1) = 0.85 \times T_{CICLO}(2) \rightarrow T_{CICLO}(2) = (1/0.85) \times T_{CICLO}(1) = 1.18 \times T_{CICLO}(1)$$

Sustituyendo $NI(2)$, $CPI(2)$, y $T_{CICLO}(2)$ en la expresión de $T_{CPU}(2)$ se tiene que:

$$\begin{aligned} T_{CPU}(2) &= CPI_2 \times NI_2 \times T_{CICLO}(2) = 3.43 \times 0.7NI_1 \times 1.18T_{CICLO}(1) \\ &= 2.83 \times NI_1 \times T_{CICLO}(1) \end{aligned}$$

Así, como $T_{CPU}(1) > T_{CPU}(2)$ se tiene que ALT2 mejora a la ALT1. Es decir, un repertorio con instrucciones más complejas, al reducir el número de instrucciones de los programas, puede mejorar las prestaciones aunque al implementar esta opción la mayor

complejidad del procesador lo haga algo más lento. Sin embargo, esto no tiene que ser así en todos los casos. Depende de la reducción en el número de instrucciones que se produzca, y de la velocidad con que el nuevo procesador ejecute las instrucciones, tal y como se ilustra en el siguiente problema.

Problema 3. Qué ocurriría en el caso de las alternativas ALT1 y ALT2 del problema anterior: (a) si hubiera solamente un 20% de instrucciones BRANCH en ALT1, y (b) si el tiempo reloj de ciclo de reloj de ALT2 fuese un 45% mayor que el de ALT1. (NOTA: solo cambia la condición que se indica en cada nuevo caso, las condiciones de las que no se dice nada se mantienen iguales a las indicadas en el problema anterior).

Solución

Compararemos las ALT1 y ALT2 para las nuevas condiciones que se indican en los apartados (a) y (b).

Solución para el apartado (a). Las Tablas 3 y 4 muestran, respectivamente, las nuevas situaciones en cada una de las alternativas ALT1 y ALT2

Tabla 3. Situación de la ALT1 en (a)

Tipo de Instrucción	Número de Instrucciones	Ciclos de cada tipo de instrucción
BRANCH	0.2 NI_1	4
Resto	0.8 NI_1	3
	$NI_1=0.2NI_1+0.8NI_1$	$CPI_1=0.2\times 4+0.8\times 3=3.2$

Tabla 4. Situación de la ALT2 en (b)

Tipo de Instrucción	Número de Instrucciones	Ciclos de cada tipo de instrucción
COMP+BRANCH	0.2 NI_1	4
Resto	$(0.8NI_1-0.2NI_1)=0.6NI_1$	3
	$NI_2=0.2NI_1+0.6NI_1=0.8NI_1$	$CPI_2=0.25\times 4+0.75\times 3=3.25$

Por lo tanto tendremos que los nuevos valores de los tiempos medios de CPU para las alternativas ALT1 y ALT2 serán, respectivamente:

$$T_{CPU}(1) = 3.2 \times NI_1 \times T_{CICLO}(1)$$

y

$$T_{CPU}(2) = CPI_2 \times NI_2 \times T_{CICLO}(2) = 3.25 \times 0.8NI_1 \times 1.18T_{CICLO}(1) \\ = 3.068 \times NI_1 \times T_{CICLO}(1)$$

En este caso, se sigue obteniendo mejora, aunque es menor que la obtenida anteriormente. Si se compara lo sucedido con la alternativa ALT2 en esta situación (a) en relación con lo que ocurría en el problema anterior es que el porcentaje de instrucciones COMP+BRANCH que se introducen es menor. Al tratarse de las instrucciones más lentas

(necesitan 4 ciclos) el incremento que se produce en CPI es menor (3.25 frente a 3.43) y aunque hay una menor reducción en el tamaño de los códigos a ejecutar (ahora los códigos tienen $0.8 \times NI_1$ en lugar de $0.7 \times NI_1$) el producto del CPI por el número de instrucciones sigue siendo lo suficientemente bajo como para compensar el incremento en el tiempo de ciclo de reloj ($3.25 \times 0.8 = 2.6$ es mayor que $3.43 \times 0.7 = 2.4$ que se tenía en ALT2, pero el valor $2.6 \times 1.18 = 3.068$ que se tiene ahora en ALT2 sigue siendo menor que el valor de 3.2 que se tiene en ALT1).

Solución para el apartado (b). En este caso, no cambian las condiciones relacionadas con el número de instrucciones de cada tipo que se tiene, lo único que sucede es que el tiempo de ciclo del procesador en ALT2 es un 45% más alto, en lugar de un 18%. Por lo tanto:

$$T_{CICLO}(2) = 1.45 \times T_{CICLO}(1)$$

y, sustituyendo, se tendría que:

$$\begin{aligned} T_{CPU}(2) &= CPI_2 \times NI_2 \times T_{CICLO}(2) = 3.43 \times 0.7NI_1 \times 1.45T_{CICLO}(1) \\ &= 3.48 \times NI_1 \times T_{CICLO}(1) \end{aligned}$$

Dado que se tenía que

$$T_{CPU}(1) = 3.3 \times NI_1 \times T_{CICLO}(1)$$

en este caso ALT2 no mejora a ALT1. Es decir, la reducción en el número de instrucciones que habría que ejecutar no compensa el ligero incremento que se produce en el tiempo medio de ciclos por instrucción (de 3.3 a 3.43) y el incremento en el tiempo de ciclo (un 45% más lento en lugar de un 18% más lento). Se puede extraer como conclusión de este problema y del anterior que, aunque una estrategia pueda ser razonable para mejorar las prestaciones, el resultado final depende de las condiciones en las que se aplique (perfil de instrucciones de los programas que ejecuten) y de las características del diseño hardware (aunque la frecuencia a la que funcione el procesador sea menor, importa cuánto menor sea).



Problema 4. Considere un procesador no segmentado con una arquitectura de tipo LOAD/STORE en la que las operaciones sólo utilizan como operandos registros de la CPU. Para un conjunto de programas representativos de su actividad se tiene que el 40% de las instrucciones son operaciones con la ALU (con 4 CPI), el 20% LOADs (con 4 CPI), el 15% STOREs (con 3 CPI) y el 25% BRANCHs (con 4 CPI).

Además, un 20% de las operaciones con la ALU utilizan operandos en registros, que no se vuelven a utilizar. ¿Se mejorarían las prestaciones si, para sustituir ese 20% de operaciones, se añaden instrucciones con un dato en un registro y otro en memoria, de un ciclo en el CPI de los BRANCHs, pero que no afectan al ciclo de reloj?.

Solución

En primer lugar se calcula el tiempo de CPU para la situación inicial. Para ello, se tiene que:

$$T_{CPU}(1) = CPI(1) \times NI(1) \times T_{ciclo}(1)$$

donde

$$CPI(1) = 0.4 \times 4 + 0.20 \times 4 + 0.15 \times 3 + 0.25 \times 4 = 3.85$$

y, por tanto:

$$T_{CPU}(1) = 3.85 \times NI(1) \times T_{ciclo}(1)$$

En la nueva situación se tiene la siguiente distribución de instrucciones y ciclos para cada una de ellas:

Tabla 5. Situación de la nueva alternativa del problema 5

Instrucción	Número	CPI
Operaciones ALU r-r	$(0.40 - 0.20 \times 0.40) \times NI(1)$	4
LOADs	$(0.20 - 0.20 \times 0.40) \times NI(1)$	4
Operaciones ALU r-m	$0.20 \times 0.40 \times NI(1)$	5
STOREs	$0.15 \times NI(1)$	3
BRANCHs	$0.25 \times NI(1)$	5
Total	$(1 - 0.2 \times 0.4) \times NI(1) = 0.92 \times NI(1)$	4.197

En la Tabla 5 se muestra que, al introducir las nuevas instrucciones de operación con la ALU con uno de los operandos en memoria:

- Se reduce el 20% de las $0.40 \times NI(1)$ instrucciones de operación con la ALU y operandos en registros a las que las nuevas instrucciones sustituyen.
- Se reducen $0.20 \times 0.40 \times NI(1)$ LOADs, ya que según se indica en el enunciado, esos LOADs sólo están en el programa para cargar uno de los operandos de las operaciones con la ALU, que no se vuelven a utilizar nunca más.
- Hay que contabilizar las $0.20 \times 0.40 \times NI(1)$ nuevas instrucciones de operación con la ALU que se introducen en el repertorio (y que sustituyen a las correspondientes operaciones con la ALU y operandos en registros).
- Como resultado, al sumar todas las instrucciones que se tienen en la nueva situación, el número total de instrucciones se reduce (lógicamente, ya que se han ahorrado instrucciones LOADs), siendo igual a

$$NI(2) = 0.92 \times NI(1)$$

Teniendo en cuenta la nueva distribución de instrucciones y sus nuevos CPIs, se tiene que:

T.S.I.I.

$$CPI(2) = \frac{0.40 - 0.20 \times 0.40}{0.92} \times 4 + \frac{0.20 - 0.20 \times 0.40}{0.92} \times 4 + \frac{0.20 \times 0.40}{0.92} \times 5 + \frac{0.15}{0.92} \times 3 \\ + \frac{0.25}{0.92} \times 5 = 4.197$$

Como el tiempo de ciclo no varía se tiene que:

$$T_{CPU}(2) = CPI(2) \times NI(2) \times T_{ciclo}(2) = 4.197 \times 0.92 \times NI(1) \times T_{ciclo}(1)$$

Es decir,

$$T_{CPU}(2) = 3.86 \times NI(1) \times T_{ciclo}(1)$$

Se puede ver en este caso que, aunque por muy poco, $T_{CPU}(1) < T_{CPU}(2)$, y por lo tanto no se mejoran las prestaciones. Si el porcentaje de instrucciones sustituidas fuese un 40% en lugar de un 20%, en ese caso, se puede ver que $NI(2) = 0.84 \times NI(1)$

$$CPI(2) = \frac{0.40 - 0.40 \times 0.40}{0.84} \times 4 + \frac{0.20 - 0.40 \times 0.40}{0.84} \times 4 + \frac{0.40 \times 0.40}{0.84} \times 5 + \frac{0.15}{0.84} \times 3 \\ + \frac{0.25}{0.84} \times 5 = 4.31$$

y, por tanto:

$$T_{CPU}(2) = CPI(2) \times NI(2) \times T_{ciclo}(2) = 4.31 \times 0.84 \times NI(1) \times T_{ciclo}(1)$$

Ahora,

$$T_{CPU}(2) = 3.62 \times NI(1) \times T_{ciclo}(1)$$

y la segunda opción mejora a la primera. Como conclusión, se puede indicar que una determinada decisión de diseño puede suponer una mejora en el rendimiento del computador correspondiente según sean las características de las distribuciones de instrucciones en los programas que constituyen la carga de trabajo característica del computador.

Por tanto, queda clara también la importancia que, para evaluar las prestaciones de los computadores, tiene la definición de conjuntos de *benchmarks* que representen de manera fiable los porcentajes de instrucciones de cada tipo que existen en los programas.

Problema 5. La empresa DataNimbus estima que debe adquirir un nuevo computador con una velocidad pico de 100 TFLOPS para alcanzar los niveles de tiempos de respuesta requeridos en su nueva generación de algoritmos para aplicaciones Big Data. Se ha decidido configurar la máquina a base de nodos HP ProLiant SL230s Gen8. Concretamente, cada uno de estos servidores tiene dos procesadores Sandy

Bridge Intel® Xeon® E5-2670 a 2.60GHz con 8 núcleos/procesador (se puede ver en, por ejemplo, http://h18000.www1.hp.com/products/quickspecs/14231_na/14231_na.pdf): (a) ¿Cuántos nodos (servidores HP ProLiant SL230s) se necesitan para configurar la máquina de 100 TFLOPS?; (b) Clasifique el nuevo servidor que se pretende adquirir, sus nodos, sus encapsulados y sus núcleos dentro de la clasificación de Flynn y dentro de la clasificación que usa como criterio el sistema de memoria; y (c) ¿Cuál es el número máximo de operaciones de coma flotante por ciclo de cada core del Intel Xeon E5-2670?.

NOTA: La velocidad pico de cada núcleo se puede determinar a partir de la lista TOP500 (<http://www.top500.org/>), en cuya edición de Noviembre de 2012 se tiene que, por ejemplo, el equipo número 13 (el Yellowstone del National Center for Atmospheric Research), que utiliza el procesador Intel® Xeon® E5-2670 a 2.6 GHz, tiene una velocidad pico de 1,503,590 GFLOPS, y contiene 72,288 núcleos.

Solución

(a) Como sabemos, por la información que nos proporciona la lista TOP500 (y que se indica en la NOTA del problema), que la máquina Yellowstone del National Center for Atmospheric Research con el procesador Intel® Xeon® E5-2670 a 2.6 GHz tiene una velocidad pico de 1,503,590 GFLOPS y contiene 72,288 núcleos, la velocidad pico de un núcleo para operaciones en coma flotante (número máximo de GFLOPS que puede proporcionar dicho núcleo) es:

$$1,503,590 \text{ GFLOPS} / 72,288 \text{ núcleos} = 20.8 \text{ GFLOPS/núcleo}$$

Dado que hay que alcanzar $100 \text{ TFLOPS} = 100 \times 10^3 \text{ GFLOPS}$, el número de núcleos que necesitamos es $(100 \times 10^3 \text{ GFLOPS}) / (20.8 \text{ GFLOPS/núcleo}) = 4,807.7$ núcleos, es decir 4,808 núcleos.

Como el servidor HP ProLiant SL230s que se utiliza en cada nodo tiene 16 núcleos (2 microprocesadores con 8 núcleos cada uno), el número de nodos necesario sería:

$$(4,808 \text{ núcleos}) / (16 \text{ núcleos/nodo}) = 300.5 \rightarrow 301 \text{ nodos}$$

(b) Desde el punto de vista de la taxonomía de Flynn es un computador MIMD. Cada uno de los microprocesadores que hay en el nodo tiene 8 núcleos que comparten la memoria local del microprocesador y por tanto son multiprocesadores UMA. Esos dos microprocesadores se interconectan en el nodo constituyendo un multiprocesador NUMA dado que cada uno de ellos tiene su memoria principal local. Los nodos están interconectados a través de una red y configuran un computador NORMA (también suele denominársele *cluster*).

(c) Como la velocidad pico de un núcleo es igual al producto de la frecuencia de reloj y el número de instrucciones por ciclo (en este caso operaciones en coma flotante por ciclo) tenemos que, en cada núcleo:

$$20.8 \text{ GFLOPS} = (\text{Operac_Coma_Flotante/ciclo}) \times \text{Frecuencia (ciclos/s)}$$

$$20.8 \text{ GFLOPS} = (\text{Operac_Coma_Flotante/ciclo}) \times 2.6 \times 10^9 \text{ (ciclos/s)}$$

y, despejando

$$(\text{Operac_Coma_Flotante/ciclo}) = (20.8 \times 10^9 \text{ Operac_Coma_Flotante/s}) / (2.6 \times 10^9 \text{ ciclos/s}) = 8$$

Por tanto, el número máximo de operaciones en coma flotante que puede terminar el núcleo por ciclo es 8.

Problema 6. Un núcleo de procesamiento (*core*) sin segmentación de cauce tiene una arquitectura de tipo LOAD/STORE en la que las operaciones sólo utilizan como operandos registros de la microarquitectura. El núcleo se encuentra integrado en un SoC (*System on Chip*) que ejecuta aplicaciones en las que, en promedio, el 30% de las instrucciones son operaciones con la ALU (4 CPI), el 25% LOADs (4 CPI), el 15% STOREs (3 CPI) y el 20% BRANCHs (4 CPI). Se ha diseñado un nuevo compilador que permite reducir un 10% el número de instrucciones de operación con la ALU y otro 10% el de instrucciones LOAD. Si el núcleo funciona a una frecuencia de reloj de 500 MHz, indique: (a) ¿Cuál es el número de MIPS que se tienen con el compilador antiguo y con el nuevo?; (b) ¿Cuál es la ganancia de velocidad que se consigue al utilizar el nuevo compilador?; (c) ¿Qué pasaría si el nuevo compilador solo fuese capaz de reducir un 20% las instrucciones STORE, dejando igual todas las demás?.

Solución

Para resolver el apartado (a) se utilizará la expresión de los MIPS (NI es el número de instrucciones y F la frecuencia de reloj):

$$\begin{aligned} MIPS &= \frac{NI(\text{instr})}{T_{CPU}(\text{segundos}) \times 10^6} = \\ &= \frac{NI(\text{instr})}{\frac{NI(\text{instr}) \times CPI(\text{ciclos/instr}) \times T_{ciclo}(\text{segundos/ciclo}) \times 10^6}{F(\text{ciclos/segundo})}} \\ &= \frac{NI(\text{instr})}{CPI(\text{ciclos/instr}) \times 10^6} \end{aligned}$$

Con el primer compilador tenemos las condiciones que se indican en la Tabla 6:

Tabla 6. Situación de partida

Instrucción	Número	CPI
Operaciones ALU r-r	$0.40 \times NI(1)$	4
LOAD	$0.25 \times NI(1)$	4
STORE	$0.15 \times NI(1)$	3
BRANCH	$0.20 \times NI(1)$	4
Total	$NI(1)$	

En este caso

$$CPI(1) = (0.40 + 0.25 + 0.20) \times 4 + 0.15 \times 3 = 3.85$$

y por tanto

$$MIPS(1) = \frac{F(\text{ciclos/segundo})}{CPI(1)(\text{ciclos/instr}) \times 10^6} = \frac{500 \times 10^6 (\text{ciclos/segundo})}{3.85 (\text{ciclos/instr}) \times 10^6} = 129.87$$

Con el nuevo compilador tendremos:

Tabla 7. Situación con el nuevo compilador

Instrucción	Número	CPI
Operaciones ALU r-r	$0.9 \times 0.40 \times NI(1)$	4
LOAD	$0.9 \times 0.25 \times NI(1)$	4
STORE	$0.15 \times NI(1)$	3
BRANCH	$0.20 \times NI(1)$	4
Total	$0.935 \times NI(1)$	

En este caso tenemos que

$$CPI(2) = \frac{(0.36 + 0.225 + 0.20) \times 4 + 0.15 \times 3}{0.935} = 3.839$$

y por tanto

$$MIPS(2) = \frac{F(\text{ciclos/segundo})}{CPI(2)(\text{ciclos/instr}) \times 10^6} = \frac{500 \times 10^6 (\text{ciclos/segundo})}{3.839 (\text{ciclos/instr}) \times 10^6} = 130.24$$

Es claro que con las optimizaciones que aplica el nuevo compilador se consigue un número mayor de MIPS (130.24 frente a 129.87)

(a) Para resolver el segundo apartado se aplican las expresiones del tiempo de CPU en los dos casos y realizamos el cociente entre el tiempo de CPU de la opción de partida y el de la opción mejorada debido al uso del nuevo compilador:

$$S = \frac{T_{CPU}(1)}{T_{CPU}(2)} = \frac{NI(1) \times CPI(1) \times T_{CICLO}(1)}{NI(2) \times CPI(2) \times T_{CICLO}(2)}$$

Teniendo en cuenta que $T_{CICLO}(1) = T_{CICLO}(2)$, y sustituyendo, obtenemos que:

$$S = \frac{T_{CPU}(1)}{T_{CPU}(2)} = \frac{NI(1) \times CPI(1)}{NI(2) \times CPI(2)} = \frac{NI(1) \times 3.85}{0.935 \times NI(1) \times 3.839} = 1.072$$

Es decir, se ha conseguido un 7.25% de mejora en las prestaciones.

(b) La situación en este caso vendría dada por la Tabla 8

Tabla 8. Situación de partida

Instrucción	Número	CPI
Operaciones ALU r-r	$0.40 \times NI(1)$	4
LOAD	$0.25 \times NI(1)$	4
STORE	$0.8 \times 0.15 \times NI(1)$	3
BRANCH	$0.20 \times NI(1)$	4
Total	$0.97 \times NI(1)$	

En este caso tenemos que

$$CPI(2) = \frac{(0.40 + 0.25 + 0.20) \times 4 + 0.8 * 0.15 * 3}{0.97} = 3.87$$

y por tanto

$$MIPS(2) = \frac{F(\text{ciclos/segundo})}{CPI(1)(\text{ciclos/instr}) \times 10^6} = \frac{500 \times 10^6 (\text{ciclos/segundo})}{3.87 (\text{ciclos/instr}) \times 10^6} = 129.19$$

Ahora, con las optimizaciones que aplica el nuevo compilador, se consigue un número menor de MIPS (129.19 frente a 129.87), a pesar de que los nuevos programas tardarían menos dado que sigue existiendo ganancia de velocidad:

$$S = \frac{T_{CPU}(1)}{T_{CPU}(2)} = \frac{NI(1) \times CPI(1)}{NI(2) \times CPI(2)} = \frac{NI(1) \times 3.85}{0.97 \times NI(1) \times 3.87} = 1.0256$$

Es decir, se ha conseguido un 2.56% de mejora en las prestaciones (algo menos que en el caso anterior). Eso significa que el tiempo de ejecución de los programas con el nuevo compilador es menor que en el caso de partida. Es lógico porque lo que consigue el compilador, tanto en este caso como en el caso anterior, es reducir el número de instrucciones que se ejecutan (y no hay cambios en el repertorio de instrucciones ni en la frecuencia de reloj, solo disminuye el número de instrucciones).

Sin embargo, los MIPS que se tienen ahora han disminuido respecto al caso de partida: los MIPS pueden reducirse a pesar de que las prestaciones aumenten. Esto se debe a que los MIPS miden lo rápido que se ejecutan las instrucciones, más que el tiempo de ejecución. Se pueden ejecutar instrucciones muy rápidamente, pero si hay que ejecutar más instrucciones, el tiempo puede ser mayor en el caso de un valor de MIPS mayor. Así, en la primera mejora, se reduce el número de instrucciones más lentas (4 ciclos) y el porcentaje de instrucciones más rápidas que se ejecutan en los nuevos programas es mayor. Por eso los MIPS también aumentan con el aumento de las prestaciones. En el segundo caso, correspondiente a la pregunta (c), se eliminan instrucciones más rápidas (con 3 ciclos), aumenta el porcentaje de instrucciones lentas en los códigos, por lo que los MIPS se reducen a pesar de que sigue existiendo mejora de prestaciones.

Problema 7. Un procesador superescalar de 64 bits a 1 GHz capaz de finalizar tres instrucciones por ciclo ejecuta el programa que se indica a continuación:

(1) start: ld	f0, a	// f0=a
(2)	add r8, r0, r2	// r8=r2 (r0=0)
(3)	addi r6, r8, #2048	// r6=r8+2048
(4)	add r12, r0, r4	// r12=r4 (r0=0)
(5) loop: ld	f2, 0(r8)	// f2=m(r8)
(6)	multd f2, f0, f2	// f2=f0*f2
(7)	ld f4, 0(r12)	// f4=m(r12)
(8)	addd f4, f2, f4	// f4=f2+f4
(9)	sd 0(r12), f4	// m(r12)=f4
(10)	addi r8, r8, #8	// r8=r8+8
(11)	addi r12, r12, #8	// r12=r12+8
(12)	sub r16, r6, r8	// r16=r6-r8
(13)	bnez r16, loop	// Si r16!=0 salta

En el programa, a es un número real, r0 es un registro que siempre está a cero, r2 contiene la dirección a partir de la cual empieza un array, X, de números reales de 64 bits, y r4 contiene la dirección a partir de la que empieza otro array también de números reales de 64 bits. ¿Qué hace el programa? ¿Cuál es el tiempo mínimo que tardaría en ejecutarse?

Solución

El programa se encarga de calcular el producto de un escalar (el número a) por un array/vector $X[i]$ ($i=1, \dots, N$) y sumarlo a un array/vector $Y[i]$ ($i=1, \dots, N$). Para ello:

- La instrucción (1) carga el escalar a en el registro f0; en la instrucción (2) el registro r8 se carga con el valor de r2 y por lo tanto apunta al comienzo del array X. Después, la instrucción (3) carga en r6 la suma de r8 y 2048 y la instrucción (4) carga en r12 la dirección de comienzo del array Y, que estaba en r4.
- Despues empieza el bucle, en la línea (5) donde está la etiqueta loop. Las instrucciones (5) a (9), respectivamente, realizan la carga de un elemento de X a f2, lo multiplican por f0 que es donde está el número a, cargan en f4 un elemento de Y y le suman el resultado de la multiplicación que está en f2, para posteriormente almacenar el resultado en la posición de memoria de la que se leyó el correspondiente elemento de Y. Por lo tanto las instrucciones (5) a (9) realizan el cálculo $Y[i] = a * X[i] + Y[i]$ donde $X[i]$ e $Y[i]$ están almacenados en memoria.
- A continuación, se incrementan los registros r8 y r12 para que apunten, respectivamente, al siguiente componente del array X (instrucción (10)) y al siguiente

componente del array Y (instrucción (11)). Dado que los números son de 64 bits habrá que restar 8 de r8 y r12 ya que 64 bits son 8 bytes y las direcciones de memoria apuntan a bytes.

- Para terminar el bucle, la instrucción (12) resta r6 y r8 y pone el resultado en r16. Así, la instrucción (13) comprueba si r16 es igual a cero. En el caso de que r16 sea igual a cero no se produce el salto a la etiqueta loop y acaba el programa. Eso quiere decir que el número de iteraciones que se hacen es igual al número que añadíamos a r8 en la instrucción (3), es decir 2048, dividido por 8 dado que en cada iteración r8 se incrementa precisamente en 8. Por tanto el número de iteraciones (es decir el valor de N) es:

$$N = \frac{2^{11}}{2^3} = 2^8 = 256$$

El tiempo mínimo de ejecución de este programa se puede estimar teniendo en cuenta que el número de instrucciones que se ejecutan es $NI = 4 + 256 \times 9 = 2,308$

Por otro lado, el número máximo de instrucciones por ciclo que puede terminar el procesador es 2 ($IPC=2$) y por lo tanto el número mínimo de ciclos por instrucción sería $CPI = 1/IPC = 1/2 = 0.5$

Teniendo en cuenta que la frecuencia de reloj es 1 GHz = 10^9 ciclos/segundo, el tiempo de ciclo sería $T_{ciclo} = 10^{-9}$ segundos/ciclo, lo que es lo mismo, 1 nanosegundo.

Teniendo en cuenta todo:

$$T_{CPU}(\text{min}) = NI \times CPI \times T_{ciclo} = 2,308 \text{ (inst)} \times 0.5 \left(\frac{\text{ciclos}}{\text{inst}} \right) \times 1 \left(\frac{\text{ns}}{\text{ciclo}} \right) = 1,154 \text{ ns}$$

Problema 8. La Figura 2 muestra un diagrama de ciclos correspondiente a la ejecución de una secuencia de instrucciones en un procesador segmentado. Cada una de las instrucciones pasa por las etapas de captación (IF), decodificación (ID), ejecución (intEX o fmulEX según la instrucción), acceso a memoria (MEM) y almacenamiento de resultado en registros (WB). También se indican los ciclos de espera de resultados previos (R-Stall) o de liberación de etapas (Stall).

Si el procesador utiliza un reloj a 500 MHz, indique cuál es el número medio de ciclos por instrucción para la secuencia de instrucciones ejecutada.

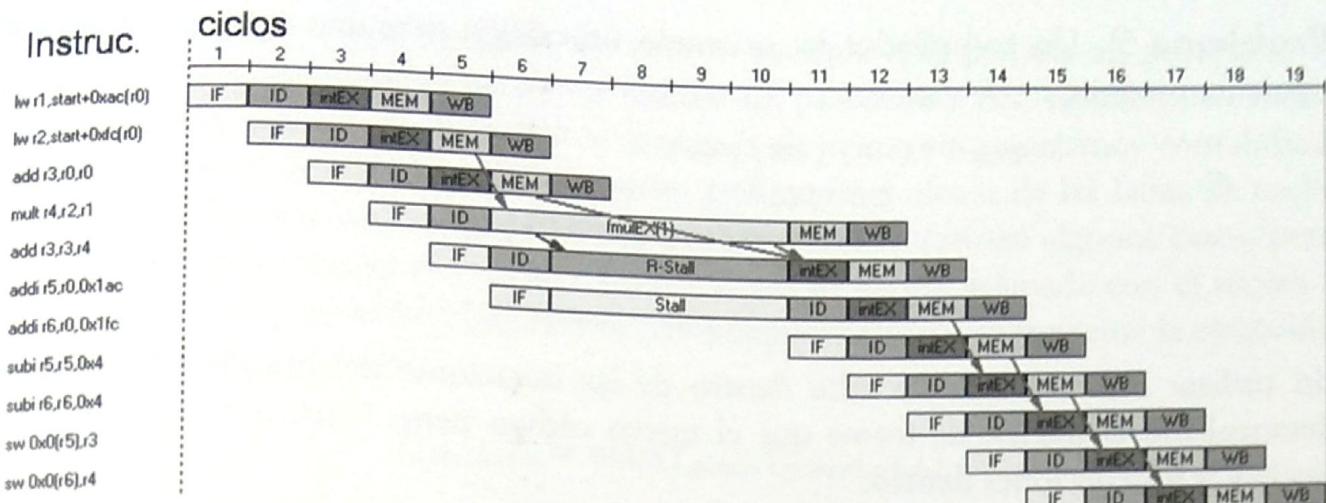


Figura 2. Ejecución de una secuencia de instrucciones en un procesador segmentado

Solución

La secuencia de instrucciones tarda 19 ciclos en ejecutarse en el procesador segmentado. Teniendo en cuenta que la frecuencia de reloj es de 500 MHz, el tiempo de ciclo es igual a:

$$T_{ciclo} = \frac{1}{Frecuencia} = \frac{1}{500 \times 10^6 \left(\frac{\text{ciclos}}{\text{seg}} \right)} = 2 \times 10^{-9} \left(\frac{\text{seg}}{\text{ciclo}} \right) = 2 \text{ nseg}$$

El tiempo que tarda en ejecutarse la secuencia de instrucciones es $T_{CPU} = 19 \times 2 = 38$ nseg. y, como el número de instrucciones que se ejecuta es $NI = 11$, dado que

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

se puede obtener el valor de CPI que se pide:

$$CPI = \frac{T_{CPU}}{NI \times T_{ciclo}} = \frac{38 \text{ nseg}}{11 \text{ inst} \times 2 \left(\frac{\text{nseg}}{\text{ciclo}} \right)} = 1.73 \left(\frac{\text{ciclos}}{\text{inst}} \right)$$

Es decir, las instrucciones tardan un promedio de 1.73 ciclos en ejecutarse. Hay que tener en cuenta que en un procesador segmentado, en el caso ideal, se terminaría de ejecutar una instrucción cada ciclo, y por tanto tendríamos que cada instrucción tardaría un ciclo. Aquí se tiene que se ejecuta menos de una instrucción por ciclo ($1/1.73=0.58$). Este funcionamiento por debajo del caso ideal se puede comprobar en la Figura 2: en los ciclos 1 a 4 no se terminan instrucciones (debido al tiempo de latencia de inicio hasta que termina de ejecutarse la primera instrucción), y en los ciclos 8 a 11 debido a la dependencia entre las instrucciones de multiplicación (que necesita más de un ciclo para ejecutarse) y la de suma que le sigue.

Por otra parte, hay que tener en cuenta que hasta el ciclo (5) no termina ninguna instrucción. Es el denominado tiempo de latencia de inicio (necesario para llenar el cauce).

Problema 9. Un compilador ha generado un código máquina optimizado para el siguiente programa:

```
for (i=0; i<N; i++)
    if ((i%2) == 0)
        par=par+c*x[i];
    else
        impar=impar-c*x[i];
```

sin utilizar instrucciones de salto dentro de las iteraciones del bucle (es decir, se ha desenrollado el bucle), de forma que el nuevo código tiene 7 instrucciones fuera del bucle y 9 instrucciones dentro.

El computador donde se ejecuta dispone de un procesador superescalar de 32 bits a 2 GHz capaz de terminar dos instrucciones por ciclo (que pueden ser instrucciones que ejecutan operaciones de coma flotante), con dos cachés internas (una para datos y otra para instrucciones) de 512 KBytes cada una, mapeo directo, política de actualización de postescritura, líneas de 32 bytes, y latencia de un ciclo de reloj. La memoria principal del computador tiene una latencia de 32 nanosegundos y ciclos burst 8-1-1-1 para el acceso a través de un bus de memoria de 250 Mhz. (a) ¿Cuántos GFLOPS pico tiene el procesador? (b) ¿Cuál es el tiempo mínimo que tarda en ejecutarse el programa para $N=2^{11}$? (c) ¿Cuántos GFLOPS alcanza en este programa?

(NOTA: Considere la situación más favorable para los datos en memoria y caché, que sea compatible con las características de la máquina y el programa; las variables par, impar, c, y x[] son números de 32 bits en coma flotante).

Solución

El programa con desenrollado sería:

```
for (i=0; i<N; i=i+2) {
    par=par+c*x[i];
    impar=impar-c*x[i+1];
}
```

por lo que el número de iteraciones es $\text{Iteraciones} = N / 2 = 2^{11}/2 = 2^{10}$ y, como se nos indica que el programa en ensamblador tiene 7 instrucciones fuera del bucle y 9 dentro, el número de instrucciones ejecutadas:

$$NI = 9 \times 2^{10} + 7 = 9,223$$

Se pasa ahora a responder a cada una de las preguntas del problema.

(a) La velocidad en GFLOPS pico del procesador, dado que nos dicen en el enunciado que el procesador puede terminar 2 operaciones de coma flotante por ciclo como máximo, es:

$$GFLOPS_{pico} = 2 \left(\frac{\text{op. coma. flotante}}{\text{ciclo}} \right) \times 2 \times 10^9 \left(\frac{\text{ciclos}}{\text{s}} \right) \times \left(\frac{1}{10^9} \right) = 4$$

(b) Podemos hacer una estimación del tiempo mínimo que tardaría en procesarse el programa teniendo en cuenta no solo el trabajo del procesador. Así, teniendo en cuenta la información que se nos da sobre la jerarquía de memoria podríamos contabilizar también el efecto de los accesos a memoria, evaluando el efecto de las faltas de caché en el procesador, que tendría que detenerse para esperar el dato en algunos casos, pero en otros podría continuar ejecutando instrucciones de forma solapada con el acceso a memoria. Por ello, una buena estimación del tiempo mínimo que consume la ejecución del programa, $T_{ejecucion}$, sería:

$$T_{ejecucion} = \max(T_{mem}, T_{procesador})$$

donde T_{mem} es una estimación del tiempo de acceso a los datos en memoria y $T_{procesador}$ el tiempo mínimo que tardaría el procesador en ejecutar el programa. La Figura 3 ilustra este razonamiento.

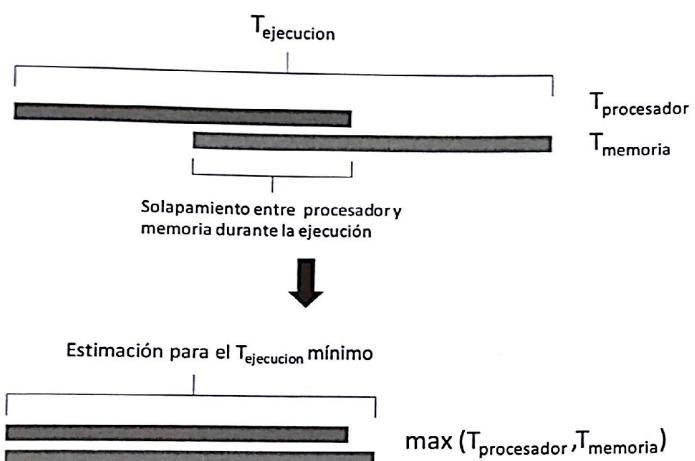


Figura 3. Estimación del tiempo mínimo de ejecución del programa

Ahora procedemos a calcular cada uno de los tiempos:

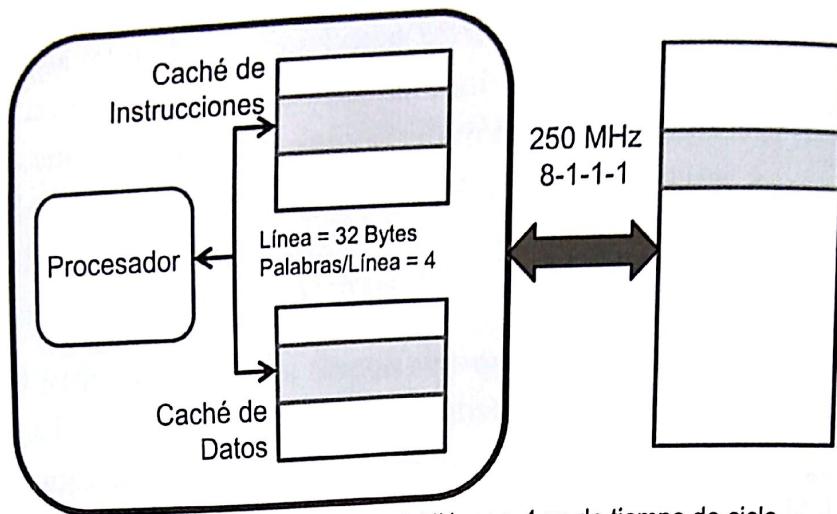
$$\begin{aligned} T_{procesador} &= NI \times CPI \times T_{ciclo} = 9,223 \text{ (inst.)} \times (1/2) \left(\frac{\text{ciclos}}{\text{inst.}} \right) \times 0.5 \left(\frac{\text{nseg}}{\text{ciclo}} \right) \\ &= 2,305.75 \text{ nseg} \end{aligned}$$

donde se ha tenido en cuenta que el procesador puede terminar dos instrucciones por ciclo como máximo, y que la frecuencia de reloj es de 2 GHz, lo que significa un tiempo de ciclo de 0.5 nanosegundos.

En cuanto a la estimación del tiempo de acceso a los datos de memoria, la Figura 4 resume las características principales de la jerarquía de cachés y memoria principal del computador.

Los accesos a memoria para cargar el código se desprecian porque el código ocupa sólo dos líneas de caché: $7 \times 4 \text{ bytes/inst.} + 9 \times 4 \text{ bytes/inst.} = 28 + 36 = 64 \text{ bytes}$ y si se divide por 32 bytes/línea obtenemos que $64 \text{ Bytes} / 32 \text{ (Bytes/línea)} = 2 \text{ líneas para el código}$.

Se producirían, por tanto, sólo dos faltas de caché, una por cada una de las líneas que se cargarían en la caché de instrucciones y se mantendrían ahí durante la ejecución del programa.



Bus de Memoria: 250 MHz >>> 4 ns de tiempo de ciclo
 Ancho de 64 bits = 2 palabras de 32 bits = 8 Bytes
 Transferencias de bus por línea: 4 (32 Bytes/línea) 8-1-1-1

Figura 4. Características relevantes de la jerarquía de memoria

En el acceso a los datos, se producirán más fallos. De hecho, se producirán tantos fallos como líneas ocupe el array $x[]$ almacenado en memoria, ya que una vez que se cargue una línea, se accede a todos los datos de esa línea de forma consecutiva. También supondremos que puesto que los accesos al dato c (que estaría en una línea aparte) solo darían lugar a un fallo de acceso y se puede despreciar dicho fallo, frente al número de fallos para acceder a $x[]$ (lo mismo que hemos hecho con los fallos para el acceso a la caché de instrucciones).

Por tanto, el número de fallos que se producen (como hemos dicho, igual al número de líneas que ocupa el array $x[]$) es:

$$Fallos = \frac{4 \left(\frac{\text{Bytes}}{\text{dato}} \right) \times 2^{11} \text{datos}}{2^5 \left(\frac{\text{Bytes}}{\text{línea}} \right)} = 2^8 \text{ líneas}$$

Teniendo en cuenta que se accede a 2^{11} datos, la tasa de fallos se puede calcular como:

$$\text{Tasa de Fallos } (1 - a) = \frac{\text{Fallos}}{\text{Accesos}} = \frac{2^8}{2^{11}} = \frac{1}{8} = 0.125$$

y la tasa de aciertos es $a = 7/8 = 0.875$.

A partir de los datos de la jerarquía de memoria que se nos da en el enunciado y se resumen en la Figura 4 se puede calcular el tiempo medio de acceso a un dato:

$$t_{memoria} = a \times t_{cache} + (1 - a) \times t_{memoria.principal} = 0.875 \times 0.5 + 0.125 \times (0.5 + 44) = 6.0 \text{ nseg.}$$

donde se ha considerado que el tiempo de acceso a caché es de un ciclo del procesador (es decir 0.5 nseg.) y que cuando hay que acceder a memoria se emplea un ciclo burst y que el bus de memoria es de 250 MHz y, por tanto tiene un ciclo de 4 nsegundos, hemos añadido los 0.5 nanosegundos que costaría llevar el dato al procesador desde la caché (estos 0.5 nseg. se podrían no tener en cuenta si se supone que el dato se pasa directamente al procesador cuando se trae desde memoria principal).

Considerando que se tienen 2^{11} accesos a memoria, tenemos que:

$$T_{memoria} = 2^{11} \times t_{memoria} = 2^{11} \times 6.0 = 12,288 \text{ nseg.}$$

de forma que

$$T_{ejecucion} = \max(T_{mem}, T_{procesador}) = \max(12,288 \text{ nseg.}, 2305.75 \text{ nseg.}) = 12,288 \text{ nseg.}$$

(c) En cuanto a los GFLOPS que se obtienen al ejecutar el programa:

$$GFLOPS = \frac{2 \text{ (op. coma. flotante/elemento del array)} \times 2^{11} \text{ elementos}}{12288 \times 10^{-9} \text{ (segundos)} \times 10^9} = 0.33$$

Como se puede ver, el número de GFLOPS obtenido es bastante menor que los 4 GFLOPS pico que el procesador alcanzaría si funcionase a pleno rendimiento (terminando de ejecutar dos instrucciones cada ciclo de reloj).



Problema 10. Un computador tiene un procesador superescalar de 32 bits a 2 GHz capaz de terminar dos instrucciones de coma flotante por ciclo, con dos cachés L1 internas (una para datos y otra para instrucciones) de 128 KBytes cada una, mapeo directo, política de actualización de postescritura, líneas de 32 bytes, y latencia de un ciclo de reloj. La memoria principal del computador tiene una latencia de 30 ns. y ciclos burst 6-1-1-1 a través de un bus de memoria de 64 bits a 200 Mhz.

Justifique cuál de los dos códigos alternativos siguientes (CÓDIGO 1 y CÓDIGO 2) permite calcular de forma más eficiente (tiempo mínimo estimado más pequeño), para $N=2^{18}$, el valor de

$$\sum_{i=0}^{N-1} (x^2(i) - X^2) \text{ donde } X = \frac{1}{N} \left(\sum_{i=0}^{N-1} x(i) \right)$$

CÓDIGO 1:

```

S=0; R=0;
for (i=0; i<N; i++) S=S+x[i];
for (i=0; i<N; i++) R=R+x[i]*x[i];
R=R-(S*S/N2); // R es el resultado

```

CÓDIGO 2:

```

S=0; R=0;
for (i=0; i<N; i++) {
    S=S+x[i];
    R=R+x[i]*x[i];
}
R=R- (S*S/N2); // R es el resultado

```

¿Qué conclusiones puede extraer del resultado?

NOTAS:

- El CÓDIGO 1 tiene 10 instrucciones fuera de los bucles (cuatro de coma flotante, una de almacenamiento y cinco de inicialización) en el primero de los bucles hay 6 instrucciones por iteración (cuatro instrucciones para controlar el bucle, una instrucción de carga del dato y una instrucción de coma flotante) y en el segundo hay 7 instrucciones por iteración (cuatro instrucciones para controlar el bucle, una instrucción de carga del dato y dos instrucciones de coma flotante).
- El CÓDIGO 2 tiene 10 instrucciones fuera del bucle (cuatro de coma flotante, una de almacenamiento y cinco de inicialización) y dentro del bucle hay 8 instrucciones por iteración (cuatro instrucciones para controlar el bucle, una instrucción de carga del dato y tres instrucciones de coma flotante).
- Tenga en cuenta que al iniciar la ejecución de los programas las cachés están vacías, y considere la situación más favorable para la ubicación de los datos en memoria y caché (compatible con las características de la máquina y el programa). En particular, el array $x[]$ está almacenado en memoria a partir de una dirección que corresponde al inicio de un bloque o línea de caché. Los datos del array $x[]$ son datos de 32 bits en coma flotante.

Solución

Primero haremos la estimación para el CÓDIGO 1, luego para el CÓDIGO 2 y compararemos los resultados.

Estimación para el CÓDIGO 1: Dado que el tiempo debido a los accesos a memoria puede solaparse en mayor o menor grado con el tiempo de ejecución del código por parte del procesador, una buena estimación del tiempo mínimo de procesamiento del código es la que corresponde a un solapamiento total de ambos tiempos. En ese caso, el mayor de ellos es el que determina el tiempo mínimo de procesamiento.

$$T_1 \geq \max(T_{\text{memoria}}, T_{\text{procesador}})$$

Por lo tanto, el problema se podría resolver estimando el tiempo del procesador, $T_{\text{procesador}}$ y el del acceso a memoria para los datos que se necesitan, T_{memoria} . Tenemos que:

$$NI = 10 + 6 \times N + 7 \times N = 10 + 6 \times 2^{18} + 7 \times 2^{18} = 3,407,882$$

$CPI = 0.5$ (se supone que el procesador puede terminar 2 instrucciones por ciclo)
 $T_{ciclo} = 0.5 \text{ ns}$ (la frecuencia de reloj es de 2 GHz)

Por tanto:

$$T_{procesador} = NI \times CPI \times T_{ciclo} = 3,407,882 \times 0.5 \times 0.5 = 851,970.5 \text{ ns} = 0.85 \text{ ms}$$

Para estimar el tiempo de memoria se tiene en cuenta que:

Tamaño de caché = 128 KBytes = 2^{17} Bytes

Tamaño de bloque o línea de caché = 32 Bytes = 2^5 Bytes

Tamaño del Array = 2^{18} elementos $\times 2^2$ Bytes/elemento = 2^{20} Bytes

Por lo tanto, el array es mayor que la caché ($2^{20}/2^{17}=8$ veces mayor) y al leerlo se irán escribiendo nuevos bloques sobre los que se habían leído previamente. Si se vuelve a leer nuevamente el array se generarán las mismas faltas que la primera vez que se leyó.

El tiempo necesario para acceder al array (es decir para traerlo a memoria caché desde memoria principal) es igual al número de bloques (o líneas) del array multiplicado por el tiempo que se tarda en traer un bloque de memoria principal a memoria caché:

$$\begin{aligned} \text{Tiempo de lectura de bloque} &= 9 \text{ ciclos del bus de memoria} \times t_{ciclo_bus_memoria} = 9 \times 5 \text{ ns} \\ &= 45 \text{ ns} \end{aligned}$$

Esto se debe a que cada bloque consta de 32 bytes, y se accede a él a través de un ciclo burst (6-1-1-1) mediante el que se realizan cuatro transferencias de 64 bits (8 bytes $\times 4 = 32$ bytes que tiene la línea). En el ciclo burst que nos indican (6-1-1-1), la primera transferencia tarda 6 ciclos y las tres restantes 1 ciclo (gracias a la localidad que se tiene por acceder a los elementos de un bloque, que están en posiciones consecutivas). Por lo tanto, la transferencia de una línea necesita $6+1+1+1=9$ ciclos y como el bus de memoria es de 200 MHz, el tiempo de ciclo es de 5 ns.

El número de bloques que tienen que leerse es igual al número de faltas de caché (cada vez que un acceso se realiza a un nuevo bloque que no se había traído se genera una falta). Como los bloques (que van a marcos de bloque o líneas de caché) tienen un tamaño de $32 = 2^5$ bytes, en el array hay:

$$(2^{20} \text{ Bytes/array}) / (2^5 \text{ Bytes/Bloque}) = 2^{15} \text{ Bloques/array}$$

Así, 2^{15} es el número de faltas (como se ha dicho, el número de bloques que hay que traerse de memoria principal a caché). Puesto que en el primer código hay que traerse el array dos veces (una en cada bucle), tendremos que el número de transferencias de bloques es de $2 \times 2^{15} = 2^{16}$. Por tanto:

$$\begin{aligned} T_{memoria} &= 2^{16} \text{ lecturas de bloques} \times 45 \text{ ns} \text{ (tiempo de lectura de un bloque)} = 2,949,120 \text{ ns} \\ &= 2.95 \text{ ms} \end{aligned}$$

Por lo tanto, para el primer código:

$$T_1 \geq \max(T_{memoria}, T_{procesador}) = \max(0.85 \text{ ms}, 2.95 \text{ ms}) = 2.95 \text{ ms}$$

También se podría haber estimado el tiempo de acceso a memoria considerando el tiempo medio de un acceso a memoria y multiplicando por el número de accesos, tal y como se hizo en el ejercicio 10). En este caso se tendría un tiempo algo mayor porque estaríamos contabilizando los accesos a memoria que no dan lugar a faltas y que ya estarían considerados dentro del tiempo del procesador. A continuación se ilustra la forma de hacer ese cálculo para este caso:

El número de faltas es igual 2^{18} bloques que tiene el array multiplicado por 2 dado que hay que acceder dos veces al mismo (una en cada bucle). Además, el número de accesos a datos es 2^{18} multiplicado por 2 bucles. Por lo tanto la tasa de fallos es:

$$(1-a) = 2^{16}/2^{19} = 1/8 = 0.125$$

y la tasa de aciertos:

$$a = 1 - 0.125 = 0.875$$

Así:

$$t_{memoria} = t_{caché} \times a + t_{MP} \times (1-a) = 0.5 \times 0.875 + (45 + 0.5) \times 0.125 = 6.125 \text{ ns}$$

Teniendo en cuenta el número de accesos totales (2^{19}):

$$T_{memoria} = 6.125 \text{ ns} \times 2^{19} = 3,211,264 \text{ ns} = 3.21 \text{ ms}$$

Como ya se ha indicado, en este caso, la estimación del tiempo de memoria es algo mayor porque aquí estamos contabilizando también el tiempo de acceso a los datos cuando están en caché, y esto se ha contabilizado ya en el tiempo de procesamiento de las instrucciones. Si lo estimamos de esta forma $T_1=3.21 \text{ ms}$.

Estimación para el CÓDIGO2. Igual que antes, el tiempo mínimo de procesamiento para este segundo caso, T_2 , se puede estimar teniendo en cuenta que:

$$T_2 \geq \max(T_{memoria}, T_{procesador})$$

Para estimar el tiempo del procesador, $T_{procesador}$, del segundo código tenemos que:

$$NI = 10 + 8 \times N = 10 + 8 \times 2^{18} = 2,097,162$$

$CPI = 0.5$ (se supone que el procesador puede terminar 2 instrucciones por ciclo)
 $T_{ciclo} = 0.5 \text{ ns}$ (la frecuencia de reloj es de 2 GHz)

Por tanto:

$$T_{procesador} = NI \times CPI \times T_{ciclo} = 2,097,162 \times 0.5 \times 0.5 = 524,290.5 \text{ ns} = 0.52 \text{ ms}$$

Para obtener el tiempo de memoria se tiene en cuenta que, como en el caso anterior:

$$\text{Tamaño de caché} = 128 \text{ KBytes} = 2^{17} \text{ Bytes}$$

$$\text{Tamaño de Bloque/línea de caché} = 32 \text{ Bytes} = 2^5 \text{ Bytes}$$

$$\text{Tamaño del Array} = 2^{18} \text{ elementos} \times 2^2 \text{ Bytes/elemento} = 2^{20} \text{ Bytes}$$

Por lo tanto, el array es mayor que la caché ($2^{20}/2^{17}=8$ veces mayor), y al leerlo se irán escribiendo nuevos bloques sobre los que se habían leído previamente. No obstante, en este caso hay que tener en cuenta que solo hay que leer el array una vez.

Así, como se hizo para la primera opción:

$$\begin{aligned} \text{Tiempo de lectura de bloque} &= 9 \text{ ciclos del bus de memoria} \times t_{\text{ciclo_bus_memoria}} = 9 \times 5 \text{ ns} \\ &= 45 \text{ ns} \end{aligned}$$

El número de bloques que tienen que leerse es igual al número de faltas de caché (cada vez que un acceso se realiza a un nuevo bloque que no se había traído se genera una falta). Como los bloques (que van a marcos de bloque o líneas de caché) tienen un tamaño de $32 = 2^5$ bytes, en el array hay:

$$(2^{20} \text{ Bytes/array})/(2^5 \text{ Bytes/Bloque}) = 2^{15} \text{ Bloques/array}$$

Así, 2^{15} es el número de faltas o lo que es igual, el número de bloques que hay que traerse de memoria principal a caché. A diferencia de lo que ocurría en el primer código sólo hay que traerse el array una vez (hay una lectura de memoria en el único bucle). Por tanto:

$$\begin{aligned} T_{\text{memoria}} &= 2^{15} \text{ lecturas de bloques} \times 45 \text{ ns (tiempo de lectura de un bloque)} = 1,474,560 \text{ ns} \\ &= 1.47 \text{ ms} \end{aligned}$$

Así, para el segundo código:

$$T_2 \geq \max (T_{\text{memoria}}, T_{\text{procesador}}) = \max (0.52 \text{ ms}, 1.47 \text{ ms}) = 1.47 \text{ ms}$$

Igual que en la primera alternativa, también se podría haber estimado el tiempo de acceso a memoria considerando el tiempo medio de un acceso a memoria y multiplicando por el número de accesos, tal y como se muestra a continuación.

Ahora el número de faltas es igual 2^{15} bloques. Es la mitad que en la primera alternativa, pero también el número de accesos a datos es la mitad: 2^{18} (hay una instrucción de acceso a memoria en cada iteración del bucle. Por lo tanto la tasa de fallos es la misma que en el caso anterior:

$$(1-a) = 2^{15}/2^{18} = 1/8 = 0.125$$

y la tasa de aciertos también será igual:

$$a = 1 - 0.125 = 0.875$$

Así:

$$t_{memoria} = t_{caché} \times a + t_{MP} \times (1-a) = 0.5 \times 0.875 + (45 + 0.5) \times 0.125 = 6.125 \text{ ns}$$

y teniendo en cuenta el número de accesos totales (que ahora es 2^{18}):

$$T_{memoria} = 6.125 \text{ ns} \times 2^{18} = 1,605,632 \text{ ns} = 1.61 \text{ ms}$$

Como antes, la estimación del tiempo de memoria de esta forma da un resultado algo mayor pues también incluimos el tiempo de acceso a los datos cuando están en caché. Así, si consideremos la segunda estimación del tiempo de acceso a memoria, tendremos que $T_2 = 1.61 \text{ ms}$ (un 8.7% mayor que con la primera forma de estimar el tiempo de memoria).

Conclusión. En primer lugar, para los dos códigos se puede ver que lo que determina el tiempo final (y constituye el cuello de botella) para resolver este problema, es el tiempo de acceso a la memoria. Hay que acceder a un array de un tamaño mayor que la caché, que ocasiona un número considerable de faltas (la tasa de aciertos de la caché es de 0.875, lo que puede considerarse relativamente bajo).

Por otro lado, el tiempo estimado para la primera opción es aproximadamente igual al doble del estimado para la segunda. Esto se debe a que en el primer caso se realizan dos accesos al array, y al tener este un tamaño varias veces el de la caché, posteriores accesos al array volverán a generar faltas porque los primeros bloques del array se han ido sustituyendo por los que se necesitan a continuación. Por tanto, en estas situaciones es mejor hacer todas las operaciones que se puedan con los datos de los bloques que se vayan trayendo de memoria principal a memoria caché. Es lo que se hace en el segundo código: incrementar la localidad de los accesos a memoria.



Problema 11. Ha aparecido en el mercado una nueva versión de un procesador en la que la única mejora con respecto a la versión anterior es una unidad de coma flotante mejorada que permite reducir el tiempo de las instrucciones de coma flotante a una cuarta parte del tiempo que consumían antes.

Suponga que en los programas que constituyen la carga de trabajo habitual del procesador las instrucciones de coma flotante consumen un promedio del 20% del tiempo del procesador antiguo: (a) ¿Cuál es la máxima ganancia de velocidad que puede esperarse en los programas si se sustituye el procesador de la versión antigua por el nuevo?. debido a nuevos aumentos en la velocidad de las operaciones en coma flotante?. (c) ¿Cuál debería ser el porcentaje de tiempo de cálculo con datos en coma flotante en los programas (en la versión antigua del procesador) para que la máxima ganancia a la que flotante obtenido en (c), ¿cuánto debería reducirse el tiempo de las operaciones en coma flotante con respecto a la situación inicial para que la ganancia máxima sea 4?.

Solución

Dado que el tiempo de las instrucciones de coma flotante se reduce a una cuarta parte, la mejora de velocidad en el recursos es $p=4$ (la inversa de la reducción del tiempo es $t/4$, siendo t el tiempo de las operaciones en coma flotante en la versión antigua del procesador).

(a) Como el porcentaje de instrucciones de coma flotante es el 20%, la fracción a la que se puede aplicar la mejora es:

$$(1-f) = 0.20, \text{ y por tanto, } f=0.80$$

Sustituyendo estos valores en la expresión de la ley de Amdahl se tiene:

$$S \leq \frac{p}{1 + f \times (p - 1)} = \frac{4}{1 + 0.80 \times (4 - 1)} = \frac{4}{1 + 2.40} = 1.17$$

(b) El valor de la máxima ganancia se obtiene calculando el límite cuando la mejora del recurso (en este caso la ganancia, o mejora en velocidad, de la unidad en coma flotante) tiende a infinito:

$$S_{\max(f=cte)} = \lim_{p \rightarrow \infty} \frac{p}{1 + f \times (p - 1)} = \frac{1}{f} = \frac{1}{0.80} = 1.25$$

Así, si por ejemplo quisiésemos obtener una ganancia de 2 tendríamos que debería verificarse que

$$2 = \frac{p}{1 + 0.80 \times (p - 1)}$$

y, despejando tendríamos $2+1.6(p-1) = p$, es decir $0.6p = -0.4$. Sería imposible de conseguir porque p tendría que ser menor que cero, $p = -0.4/0.6$ (p es un cociente de tiempos y siempre es positivo).

(c) Para que la ganancia máxima (es decir cuando la mejora, p , tiende a infinito) sea igual a 8, la fracción de tiempo que no se reduciría con la mejora se obtendría a partir de:

$$S_{\max(f=cte)} = \frac{1}{f} = 8$$

Por lo que $f=1/8=0.125$, y la fracción de tiempo correspondiente a operaciones en coma flotante (que es donde podría aprovecharse la mejora) tendría que ser $(1-f)=1-0.125=0.875$. Es decir, al menos un 87.5% del tiempo de ejecución secuencial debería deberse a las operaciones en coma flotante.

(d) Si, al utilizar el valor de f anterior, es decir $f=0.125$, se deseara obtener una mejora de velocidad de 4 en los programas, la mejora de velocidad en las operaciones de coma flotante (es decir p) se puede obtener a partir de la ley de Amdahl:

$$S \leq 4 = \frac{1}{0.125 + \frac{0.875}{p}}$$

Por lo que $0.5 + (3.5/p) = 1$, y $p = 3.5/0.5 = 7$. Es decir, las operaciones en coma flotante deberían ser siete veces más rápidas.

Problema 12. En un programa que se ejecuta en un procesador no segmentado que funciona a 200 MHz, hay un 25% de instrucciones LOAD que necesitan 4 ciclos, un 15% de instrucciones STORE que necesitan 3 ciclos, un 40% de instrucciones con operaciones en coma flotante que necesitan 8 ciclos, y un 20% de instrucciones de salto que necesitan 4 ciclos. (a) Si en las instrucciones de coma flotante, el tiempo debido a la operación supone 4 ciclos (el resto de ciclos de la instrucción corresponden a la captación, decodificación y acceso a los datos, y almacenamiento de resultados) determine cuál es la ganancia que se puede obtener si se mejora el diseño de la unidad que realiza las operaciones en coma flotante y se reduce su tiempo a la mitad de ciclos. (b) ¿Cuál es la máxima ganancia de velocidad que se puede conseguir por mejoras en la unidad que ejecuta las operaciones en coma flotante?

Solución

(a) Las características del programa que describe el problema se muestra en la Tabla 9.

Tabla 9. Situación de partida

Instrucción	Número	CPI
Coma flotante	$0.40 \times NI_1$	8
LOAD	$0.25 \times NI_1$	4
STORE	$0.15 \times NI_1$	3
BRANCH	$0.20 \times NI_1$	4
Total	NI_1	

El tiempo de partida es

$$T_{CPU}(1) = CPI_1 \times NI_1 \times T_{CICLO}(1)$$

donde, $T_{CICLO}(1) = 1/(200 \times 10^6) = 5 \times 10^{-9} \text{ seg} = 5 \text{ nseg}$, y el valor de CPI_1 se obtiene sumando, para todos los tipos de instrucciones, los productos del número de ciclos que necesita cada instrucción y la frecuencia con que aparece dicha instrucción:

$$CPI_1 = \left(\frac{0.4 \times NI_1}{NI_1} \times 8 \right) + \left(\frac{(0.25 \times NI_1) + (0.20 \times NI_1)}{NI_1} \times 4 \right) + \left(\frac{0.15 \times NI_1}{NI_1} \times 3 \right) = 5.85$$

y, por tanto

$$T_{CPU}(1) = 5.85 \times NI_1 \times 5 = 29.25 \times NI_1 \text{ nseg}$$

A continuación se calcula el tiempo medio de CPU para la mejora, teniendo en cuenta que, dado que el tiempo dedicado al cálculo de la operación en coma flotante pasaría de 4 ciclos a 2 ciclos, el *CPI* de las instrucciones en coma flotante pasaría de 8 ciclos a 6 ciclos y, así:

$$CPI_2 = \left(\frac{0.4 \times NI_1}{NI_1} \times 6 \right) + \left(\frac{(0.25 \times NI_1) + (0.20 \times NI_1)}{NI_1} \times 4 \right) + \left(\frac{0.15 \times NI_1}{NI_1} \times 3 \right) = 5.05$$

Dado que ni el número de instrucciones ni la frecuencia de reloj cambian:

$$T_{CPU}(2) = CPI_2 \times NI_2 \times T_{CICLO}(2) = 5.05 \times NI_1 \times 5 = 25.25 \times NI_1 \text{ nseg}$$

De esta forma, la ganancia obtenida es:

$$S = \frac{T_{CPU}(1)}{T_{CPU}(2)} = \frac{29.25}{25.25} = 1.16$$

(b) La máxima ganancia de velocidad que se podría alcanzar al reducir el tiempo de las operaciones en coma flotante se produciría cuando el tiempo que se consumiera en estas operaciones fuese igual a cero (algo que no sería físicamente posible, pero se trata de obtener la máxima ganancia por este tipo de mejoras). En esta situación, las instrucciones de coma flotante necesitarían 4 ciclos en lugar de los 8 ciclos de partida, con lo que

$$CPI_{min} = \left(\frac{0.4 \times NI_1}{NI_1} \times 4 \right) + \left(\frac{(0.25 \times NI_1) + (0.20 \times NI_1)}{NI_1} \times 4 \right) + \left(\frac{0.15 \times NI_1}{NI_1} \times 3 \right) = 4.25$$

y el tiempo de CPU mínimo sería (teniendo en cuenta nuevamente que no cambian ni el número de instrucciones ni el tiempo de ciclo):

$$T_{CPU}(min) = CPI_{min} \times NI_1 \times T_{CICLO}(1) = 4.25 \times NI_1 \times 5 = 21.25 \times NI_1 \text{ nseg}$$

y la ganancia máxima que se obtendría es:

$$S_{max} = \frac{T_{CPU}(1)}{T_{CPU}(min)} = \frac{29.25}{21.25} = 1.38$$

Esta ganancia máxima también se podría obtener a partir de la ley de Amdahl:

$$S \leq \frac{p}{1 + f \times (p - 1)}$$

donde p es la ganancia de velocidad del recurso que se ha mejorado (en este problema la unidad de operación en coma flotante), y f es la fracción del tiempo de ejecución del programa antes de la mejora donde no se puede aplicar esta mejora. Como nos piden debemos considerar que p tiende a infinito, de forma que



$$S_{max} = \lim_{p \rightarrow \infty} \frac{p}{1 + f \times (p - 1)} = \frac{1}{f}$$

y solo tendríamos que calcular el valor de f :

$$f = \frac{T_{CPU}(1)^{(no\ mejorable)}}{T_{CPU}(1)}$$

El tiempo de CPU no mejorable, $T_{CPU}^{(no\ mejorable)}$, sería el que consumen todas las etapas de las instrucciones de coma flotante menos las que corresponden a la ejecución de la propia operación en coma flotante (es decir cuatro ciclos) y el tiempo que consumen todas las demás. Así,

$$\begin{aligned} CPI_1^{(no\ mejorable)} &= \\ &= \left(\frac{0.4 \times NI_1}{NI_1} \times (8 - 4) \right) + \left(\frac{(0.25 \times NI_1) + (0.20 \times NI_1)}{NI_1} \times 4 \right) \\ &+ \left(\frac{0.15 \times NI_1}{NI_1} \times 3 \right) = 4.25 \end{aligned}$$

y el tiempo de CPU no mejorable (teniendo en cuenta que tampoco cambian ni el número de instrucciones ni el tiempo de ciclo):

$$\begin{aligned} T_{CPU}(1)^{(no\ mejorable)} &= CPI_1^{(no\ mejorable)} \times NI_1 \times T_{CICLO}(1) = 4.25 \times NI_1 \times 5 \\ &= 21.25 \times NI_1 \text{ nseg} \end{aligned}$$

por lo que obtenemos nuevamente una mejora máxima del 38%:

$$S_{max} = \frac{1}{f} = \frac{T_{CPU}(1)}{T_{CPU}(1)^{(no\ mejorable)}} = \frac{29.25}{21.25} = 1.38$$

Problema 13. En un programa que se ejecuta en un procesador no segmentado que funciona a 500 MHz, hay un 15% de instrucciones LOAD que necesitan 4 ciclos, un 5% de instrucciones STORE que necesitan 3 ciclos, un 60% de instrucciones con operaciones con la ALU que necesitan 6 ciclos, y un 20% de instrucciones de salto que necesitan 4 ciclos. Si en las instrucciones con la ALU, la operación de la ALU consume 3 ciclos, determine cuál es la ganancia que se puede obtener si se mejora el diseño de la ALU de forma que se reduce el tiempo de ejecución de las operaciones a un ciclo. ¿Cuántas instrucciones con la ALU más debería tener el programa para conseguir una ganancia igual a 1.4 con la mejora indicada? ¿Qué pasaría si se quisiera conseguir una ganancia de 1.5?

Solución

La situación de la que se parte se muestra en la Tabla 10:

Tabla 10. Situación de partida del problema 13

Instrucción	Número	CPI
Operaciones con ALU	$0.60 \times NI_1$	6
LOAD	$0.15 \times NI_1$	4
STORE	$0.05 \times NI_1$	3
BRANCH	$0.20 \times NI_1$	4
Total	NI_1	

El tiempo de partida es

$$T_{CPU}(1) = CPI_1 \times NI_1 \times T_{CICLO}(1)$$

donde, $T_{ciclo}(1) = 1/(500 \times 10^6) = 2 \times 10^{-9} \text{ seg} = 2 \text{ nseg}$, y el valor de CPI_1 se obtiene sumando, para todos los tipos de instrucciones, los productos del número de ciclos que necesita cada instrucción y la frecuencia con que aparece dicha instrucción:

$$CPI_1 = \left(\frac{0.6 \times NI_1}{NI_1} \times 6 \right) + \left(\frac{(0.15 \times NI_1) + (0.20 \times NI_1)}{NI_1} \times 4 \right) + \left(\frac{0.05 \times NI_1}{NI_1} \times 3 \right) = 5.85$$

y, por tanto

$$T_{CPU}(1) = 5.15 \times NI_1 \times 2 = 10.3 \times NI_1 \text{ nseg}$$

Con la mejora, el tiempo dedicado al cálculo de la operación con la ALU pasaría de 3 ciclos a 1 ciclos, y por tanto el CPI de las instrucciones con la ALU pasaría de 6 ciclos a 4 ciclos y, así:

$$CPI_2 = \left(\frac{0.6 \times NI_1}{NI_1} \times 4 \right) + \left(\frac{(0.15 \times NI_1) + (0.20 \times NI_1)}{NI_1} \times 4 \right) + \left(\frac{0.05 \times NI_1}{NI_1} \times 3 \right) = 3.95$$

Dado que ni el número de instrucciones ni la frecuencia de reloj cambian:

$$T_{CPU}(2) = CPI_2 \times NI_2 \times T_{CICLO}(2) = 3.95 \times NI_1 \times 2 = 7.9 \times NI_1 \text{ nseg}$$

De esta forma, la ganancia obtenida es:

$$S = \frac{T_{CPU}(1)}{T_{CPU}(2)} = \frac{10.3}{7.9} = 1.304$$

Para resolver la segunda cuestión del problema utilizaremos la ley de Amdahl, que establece que la ganancia de velocidad está acotada según se indica en la expresión:

$$S \leq \frac{p}{1 + f \times (p - 1)}$$

Así, para poder alcanzar una mejora de 1.4, tiene que verificarse que, al menos:

$$\frac{p}{1 + f \times (p - 1)} = 1.4$$

En primer lugar hay que tener en cuenta que con la mejora que se ha aplicado, las instrucciones con la ALU tardan 4 ciclos en lugar de 6. Por lo tanto, $p=6/4=1.5$, y sustituyendo en la expresión anterior, se tendría:

$$\frac{1.5}{1 + f \times 0.5} = 1.4$$

y al despejar tendríamos que $f = 0.142$ (si el valor de f fuese menor que ese valor se tendría una cota superior para la ganancia de velocidad que sería mayor que 1.4).

A continuación determinaremos el número de instrucciones que tendríamos que añadir. Para ello, consideramos la Tabla 11, donde el número de instrucciones de operaciones con la ALU que tendría el nuevo programa con respecto al inicial crece en $X \times NI_1$, (es decir, X es el porcentaje de instrucciones que se introducen, con respecto al número de instrucciones del programa inicial).

Tabla 11. Situación en el nuevo programa (respecto al inicial)

Instrucción	Número	CPI
Operaciones con ALU	$0.60 \times NI_1 + X \times NI_1$	6
LOAD	$0.15 \times NI_1$	4
STORE	$0.05 \times NI_1$	3
BRANCH	$0.20 \times NI_1$	4
Total	$NI_1 + X \times NI_1$	

Con esta situación, el tiempo de partida sería

$$T_{CPU}(2) = CPI_2 \times NI_2 \times T_{CICLO}(2) = \\ = \left(\frac{(0.6 + X) \times 6 + 0.35 \times 4 + 0.05 \times 3}{1 + X} \right) \times ((1 + X) \times NI_1) \times T_{CICLO}(1)$$

De este tiempo $T_{CPU}(2)$, la parte a la que no se le puede aplicar la mejora es la parte que corresponde a las instrucciones LOAD, STORE, y BRANCH, es decir:

$$T_{CPU}^*(2) = \left(\frac{0.35 \times 4 + 0.05 \times 3}{1 + X} \right) \times ((1 + X) \times NI_1) \times T_{CICLO}(1)$$

y por tanto, f vendrá dada por:

$$f = \frac{0.35 \times 4 + 0.05 \times 3}{(0.6 + X) \times 6 + 0.35 \times 4 + 0.05 \times 3} = \frac{1.55}{5.15 + 6 \times X} \leq 0.142$$

Por lo que, despejando se tiene que $0.819 \leq 0.852 \times X$, con lo que el número de instrucciones que hay que añadir es $X \times NI_1$, con $X \geq 0.961$. Es decir, hay que añadir casi tantas instrucciones de operación con la ALU como instrucciones teníamos en el programa inicial.

No se podría conseguir una ganancia de 1.5 dado que la mejora de velocidad de las instrucciones con la ALU es precisamente igual a 1.5. Por tanto, en la expresión de la ley de Amdahl tendríamos:

$$1.5 \leq \frac{1.5}{1 + f \times 0.5}$$

Sólo si $f=0$ se consigue la igualdad. Únicamente si todas las instrucciones son instrucciones de operación con la ALU (o tuviéramos un programa de tamaño infinito con un número infinito de operaciones con la ALU) para que f fuese igual a 0 podríamos conseguir esa ganancia de 1.5.



Problema 14. Suponga el siguiente bucle en el que los arrays $a[]$ y $b[]$, y la variable S son números reales:

```
for (i=0; i<N; i++) S=S+a[i]*b[i];
```

(a) Si debe ejecutarse en 0.50 segundos para $N=10^9$. ¿Cuántos GFLOPS debe proporcionar el programa como mínimo, suponiendo que el producto y la suma en coma flotante tienen un coste similar?. (b) Si el procesador funciona a 2 GHz ¿Cuántas operaciones en coma flotante tiene que terminar por ciclo?. (c) Suponiendo que el programa en ensamblador tiene $6 \times N + 4$ instrucciones, ¿Cuántas instrucciones por ciclo debe terminar el procesador si en el 20% de los 0.50 segundos que tarda el programa no se termina ninguna instrucción debido a accesos a memoria, por fallos de caché?. (d) Utilice la ley de Amdahl para estimar la máxima ganancia de velocidad que se podría conseguir en la ejecución de ese programa incrementando el número de instrucciones por ciclo que puede terminar el procesador.

Solución

El bucle calcula el producto escalar de dos vectores $a[]$ y $b[]$, que se carga en la variable S . El número de operaciones en coma flotante que hay que realizar es $2 \times N$ (una suma y un producto en coma flotante por iteración, que se nos dice que tienen el mismo coste).

(a) Dado que conocemos el número de operaciones en coma flotante que hay que ejecutar y el tiempo que se tarda en hacerlo, se pueden calcular los GFLOPS:

$$GFLOPS = \frac{\text{Operaciones en Coma Flotante}}{T \times 10^9} = \frac{2 \times 10^9}{0.5 \text{seg} \times 10^9} = 4$$

(b) Como el procesador funciona a 2 GHz, el número de ciclos que se completan en 0.5 segundos es igual a:

$$\text{Ciclos} = 2 \times 10^9 (\text{ciclos/seg}) \times 0.5 (\text{seg}) = 1 \times 10^9$$

Puesto que debe ejecutar 2×10^9 operaciones en coma flotante, tenemos que el número de operaciones por ciclo que el procesador debe ser capaz de terminar:

$$\frac{2 \times 10^9 (\text{op. coma. flotante})}{1 \times 10^9 (\text{ciclos})} = 2 (\text{op. coma. flotante/ciclo})$$

(c) Dado que en el 20% del tiempo de ejecución no se completan instrucciones, eso significa que las $6 \times N + 4$ instrucciones se han completado en $0.8 \times 0.5 = 0.4$ segundos. En estos 0.4 segundos, el número de ciclos que se completa es

$$\text{Ciclos} = 2 \times 10^9 (\text{ciclos/seg}) \times 0.4 (\text{seg}) = 8 \times 10^8$$

Dado que tenemos $6 \times 10^9 + 4$ instrucciones, el número de instrucciones se obtendría a partir del cociente:

$$\frac{6 \times 10^9 + 4 (\text{instrucciones})}{8 \times 10^8 (\text{ciclos})} \approx 7.5 (\text{instrucciones/ciclo})$$

Es decir, el procesador debería ser capaz de terminar 7.5 instrucciones por ciclo durante el tiempo en el que terminan instrucciones. Es decir, su microarquitectura debe ser capaz de terminar 8 instrucciones por ciclo, o más. Se trata de un número muy elevado para las capacidades usuales en los procesadores con paralelismo de instrucciones.

(d) Se puede suponer que el incremento en el número de instrucciones por ciclo no reduciría el tiempo en el que el procesador no termina instrucciones por los accesos a memoria principal que se producen (las faltas de caché serían las mismas y no se producen cambios en la jerarquía de memoria). Por lo tanto el valor de f en la ley de Amdahl sería el 20% del tiempo de ejecución inicial:

$$f = \frac{0.2 \times 0.50 \text{ seg}}{0.50 \text{ seg}} = 0.2$$

y la ganancia de velocidad sería, como mucho

$$S_{\max} = \frac{1}{f} = \frac{1}{0.2} = 5$$