2º curso / 2º cuatr.

Grado en Ing. Informática

Arquitectura de Computadores: Exámenes y Controles

Control Temas 1 a 3 (grupos C y D) del 24/05/2013 resuelto

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita

1 Enunciado Control Temas 1 a 3 del 24/05/2013

Ejercicio 1. Un compilador ha generado un código máquina optimizado para el siguiente programa

```
par=0; impar=0;
for (i=0;i<1022;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 (porque se ha usado la técnica de desenrollado el bucle del Seminario 4). El código (vea la tabla de abajo) tiene (considere que `par`, `impar`, `c`, y `x[]` son números de 32 bits en coma flotante y `x[]` comienza en una dirección múltiplo del tamaño de un bloque de memoria):

- Un número de iteraciones en el bucle de 1022/2.
- 7 instrucciones fuera del bucle: 2 de almacenamiento en memoria (STORE), 5 instrucciones para inicializar registros (`N`, `i`, `c`, `par` e `impar` estarán en registros).
- 10 instrucciones dentro del bucle:
  - 4 instrucciones para implementar el bucle for: incremento de la variable de control i (ADD), comparación (CMP), salto condicional (BRCND) y un salto incondicional (BR);
  - 4 instrucciones coma flotante (MULFP, ADDFP, SUBFP) y
  - 2 instrucciones de carga desde memoria a registro (LD).

Código optimizado	**Instrucciones máquina**
`par=0; impar=0;`	5 instr. inialización de registros (par, impar, i, N, c)

`for (i=0;i<N;i=i+2)`
 	
`par=par+c*x[i];`
`impar=impar-c*x[i+1];`
	ADD i,2
	CMP i,N
	BRCND si i>=N salto a STORE
	2 LOAD (x[i],x[i+1])
	2 MULFP, 1 ADDFP, 1SUBFP
	BR a ADD

	2 STORE (par, impar)

El computador donde se ejecuta dispone de:

1. Un procesador superescalar de 32 bits a 2 GHz capaz de terminar **una instrucción de coma flotante** cada **2 ciclos**, y **2 instrucciones** de cualquier otro tipo **por ciclo**, excepto instrucciones de carga, cuyo tiempo depende de si hay o no fallo de cache (si no lo hay, supone **1 ciclo**) y las instrucciones de **almacenamiento** que suponen **1 ciclo** por instrucción.

2. Dos caches integradas en el chip de procesamiento (una para datos y otra para instrucciones) de 512 KBytes cada una, mapeo directo, política de actualización de postescritura, líneas de cache de 32 bytes, y latencia de **1 ciclo** de reloj.

3. Una memoria principal a la que se accede a través de un bus de memoria de 200 MHz y 64 bits con ciclos burst 6-1-1-1.

Conteste a las siguientes cuestiones:

(a) ¿Cuántas instrucciones se ejecutan antes del primer acceso a memoria? ¿Qué número de ciclos de reloj tarda su ejecución? Razone/explique su respuesta

(b) ¿Qué número de ciclos de reloj supondría la ejecución de una iteración del bucle si todos los datos de memoria a los que se accede están en la memoria cache? Razone/explique su respuesta.

(c) ¿Cuántas bloques de memoria ocupa el vector `x`?¿Qué número de ciclos de reloj supondría la ejecución de una iteración del bucle si nunca hay acierto de cache cuando se accede a un nuevo bloque de `x`? Razone/explique su respuesta.

(d) En caso de producirse fallos en los acceso a todos los bloques de `x`. ¿Cuántas instrucciones se ejecutan hasta finalizar el programa desde el primer load de la última iteración del bucle? ¿Qué número de ciclos de reloj tarda la ejecución de estas instrucciones? Razone/explique su respuesta

(e) Optenga los ciclos que tardaría en ejecutarse todo el código si no hay ningún fallo de cache. Optenga los MFLOPS para este caso.

(f) Optenga los ciclos que tardaría en ejecutarse todo el código en caso de producirse fallos en los acceso a todos los bloques de `x`. Optenga los MFLOPS para este caso.

Cuestión 1. Deduzca la expresión matemática que se suele utilizar para caracterizar la ley de Gustafson. Defina claramente y sin ambigüedad el punto de partida que va a utilizar para deducir esta expresión y cada una de las etiquetas que utilice. ¿Qué nos quiere decir Gustafson con esta ley?

Cuestión 2. ¿Cuál de los siguientes modelos de consistencia permite mejores tiempos de ejecución? Justifique su respuesta.

a) modelo de ordenación débil
b) modelo implementado en los procesadores de la línea x86
c) modelo de consistencia secuencial
d) modelo de consistencia de liberación

2 Solución Control Temas 1 a 3 del 24/05/2013

Ejercicio 1. Un compilador ha generado un código máquina optimizado para el siguiente programa

```
par=0; impar=0;
for (i=0;i<1022;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 (porque se ha usado la técnica de desenrollado el bucle del Seminario 4). El código (vea la tabla de abajo) tiene (considere que `par`, `impar`, `c`, y `x[]` son números de 32 bits en coma flotante y `x[]` comienza en una dirección múltiplo del tamaño de un bloque de memoria):

- Un número de iteraciones en el bucle de 1022/2.
- 7 instrucciones fuera del bucle: 2 de almacenamiento en memoria (STORE), 5 instrucciones para inicializar registros (`N`, `i`, `c`, `par` e `impar` estarán en registros).
- 10 instrucciones dentro del bucle:
  - 4 instrucciones para implementar el bucle for: incremento de la variable de control i (ADD), comparación (CMP), salto condicional (BRCND) y un salto incondicional (BR);
  - 4 instrucciones coma flotante (MULFP, ADDFP, SUBFP) y
  - 2 instrucciones de carga desde memoria a registro (LD).

Código optimizado	**Instrucciones máquina**
`par=0; impar=0;`	5 instr. inialización de registros (par, impar, i, N, c)

`for (i=0;i<N;i=i+2)`
 	
`par=par+c*x[i];`
`impar=impar-c*x[i+1];`
	ADD i,2
	CMP i,N
	BRCND si i>=N salto a STORE
	2 LOAD (x[i],x[i+1])
	2 MULFP, 1 ADDFP, 1SUBFP
	BR a ADD

	2 STORE (par, impar)

El computador donde se ejecuta dispone de:

1. Un procesador superescalar de 32 bits a 2 GHz capaz de terminar **una instrucción de coma flotante** cada **2 ciclos**, y **2 instrucciones** de cualquier otro tipo **por ciclo**, excepto instrucciones de carga, cuyo tiempo depende de si hay o no fallo de cache (si no lo hay, supone **1 ciclo**) y las instrucciones de **almacenamiento** que suponen **1 ciclo** por instrucción.

2. Dos caches integradas en el chip de procesamiento (una para datos y otra para instrucciones) de 512 KBytes cada una, mapeo directo, política de actualización de postescritura, líneas de cache de 32 bytes, y latencia de **1 ciclo** de reloj.

3. Una memoria principal a la que se accede a través de un bus de memoria de 200 MHz y 64 bits con ciclos burst 6-1-1-1.

Conteste a las siguientes cuestiones:

(a) ¿Cuántas instrucciones se ejecutan antes del primer acceso a memoria? ¿Qué número de ciclos de reloj tarda su ejecución? Razone/explique su respuesta

(b) ¿Qué número de ciclos de reloj supondría la ejecución de una iteración del bucle si todos los datos de memoria a los que se accede están en la memoria cache? Razone/explique su respuesta.

(c) ¿Cuántas bloques de memoria ocupa el vector `x`?¿Qué número de ciclos de reloj supondría la ejecución de una iteración del bucle si nunca hay acierto de cache cuando se accede a un nuevo bloque de `x`? Razone/explique su respuesta.

(d) En caso de producirse fallos en los acceso a todos los bloques de `x`. ¿Cuántas instrucciones se ejecutan hasta finalizar el programa desde el primer load de la última iteración del bucle? ¿Qué número de ciclos de reloj tarda la ejecución de estas instrucciones? Razone/explique su respuesta

(e) Optenga los ciclos que tardaría en ejecutarse todo el código si no hay ningún fallo de cache. Optenga los MFLOPS para este caso.

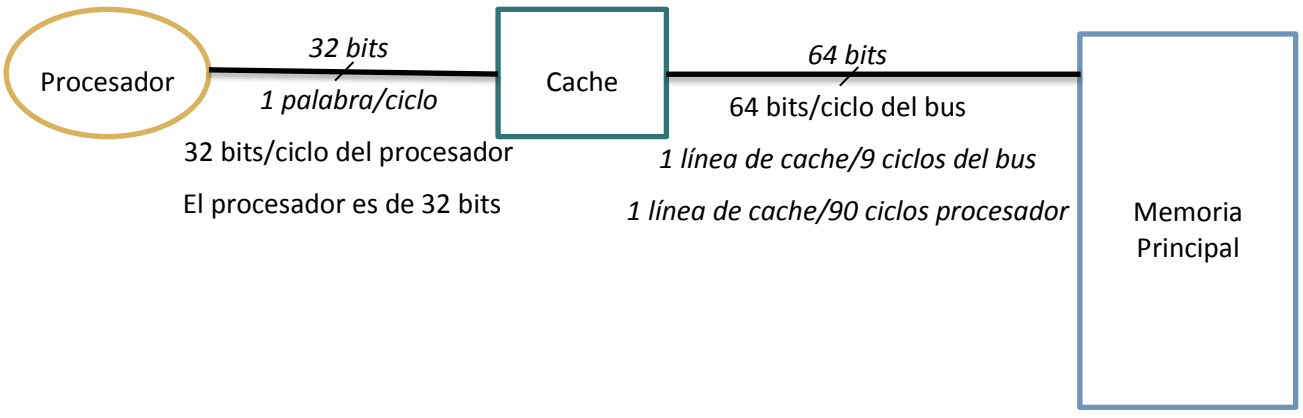
(f) Optenga los ciclos que tardaría en ejecutarse todo el código en caso de producirse fallos en los acceso a todos los bloques de `x`. Optenga los MFLOPS para este caso.

Solución

Datos del ejercicio:

Tipo i	$1/CPI_i$	CPI_i
LOAD	1 instr./ciclo	1 si NO hay fallo de cache
STORE	1 instr./ciclo	1
FP	0.5 instr./ciclo	2
Resto	2 instr./ciclo	0.5

Burst (ráfaga)	6 – 1 – 1 – 1	Frecuencia del bus = 200 MHz => ciclo de 1/200 MHz = 5 ns => 6 ciclos = 30ns
Flanco reloj	↑ ↑ ↑ ↑	Bus de 64 bits (8B): el bus trasfiere 8B (dos palabras de 4B) cada ciclo de reloj
Bytes transferidos	8B 8B 8B 8B = 32B	Línea de cache de 32B: se supone entonces que un *burst* obtiene una línea
Ciclos procesador	60 10 10 10 ciclos	Ciclos procesador por ciclos bus= $^{2\ GHz\ proc.}/_{0.2\ GHz\ bus} = 10$
Ciclos componente entre MP y cache	60 x[0] x[1] 70 x[2] x[3] 80 x[4] x[5] 90 cicl… x[6] x[7]	x comienza en un múltiplo de una línea de cache. Al acceder a x[0] hay un fallo de cache. Tras 60 ciclos estarán en la cache 8B, x[0] y x[1], (en 61 ciclos estará x[0] en el procesador y un ciclo más tarde estará x[1]), tras 10 ciclos más (70 ciclos) estarán x[2] y x[3] en la cache, tras otros 10 ciclos (80 ciclos) estarán x[4] y x[5], tras otros 10 ciclos (90 ciclos) x[6] y x[7]. A continuación, el acceso a x[8] y x[9] supone otro fallo de cache de 60 ciclos. Entonces, el acceso a una línea de cache completa supone 90 ciclos.



(a) ¿Cuántas instrucciones se ejecutan antes del primer acceso a memoria? ¿Qué número de ciclos de reloj tarda su ejecución? Razone/explique su respuesta

Antes del primer acceso a memoria se ejecutan las **8 instrucciones** que hay en verde en la siguiente tabla:

Código optimizado	Instrucciones máquina	Ciclos
`par=0; impar=0;`	5 instr. inialización de registros (par, impar, i, N, c)	5 instr. * 0.5 ciclos/instr. = 2.5 ciclos

`for (i=0;i<N;i=i+2)`
 	
`par=par+c*x[i];`
`impar=impar-c*x[i+1];`
	ADD i,2
	CMP i,N
	BRCND si i>=N salto a STORE
	2 LOAD (x[i],x[i+1])
	2 MULFP, 1 ADDFP, 1SUBFP
	BR a ADD
		0.5 ciclos
		0.5 ciclos
		0.5 ciclos (1.5 ciclos)
		2 i * 1 c/i = 2 ciclos
		4 i * 2 c/i = 8 ciclos
		0.5 ciclos

	2 STORE (par, impar)	2 instr. * 1 ciclo/instr. = 2 ciclos

En la tabla aparecen los ciclos que tarda en ejecutarse cada instrucción teniendo en cuenta los datos. Las instrucciones en verde tardan un total de **4 ciclos** en ejecutarse.

(b) ¿Qué número de ciclos de reloj supondría la ejecución de una iteración del bucle si todos los datos de memoria a los que se accede están en la memoria cache? Razone/explique su respuesta.

Como se puede ver en la siguiente tabla una iteración supone **12 ciclos** de reloj si no hay fallos de cache (las **10 instrucciones** que forman parte del ciclo aparecen en violeta):

Código optimizado	Instrucciones máquina	Ciclos
`par=0; impar=0;`	5 instr. inialización de registros (par, impar, i, N, c)	5 instr. * 0.5 ciclos/instr. = 2.5 ciclos

`for (i=0;i<N;i=i+2)`
 	
`par=par+c*x[i];`
`impar=impar-c*x[i+1];`
	ADD i,2
	CMP i,N
	BRCND si i>=N salto a STORE
	2 LOAD (x[i],x[i+1])
	2 MULFP, 1 ADDFP, 1 SUBFP
	BR a ADD
		0.5 ciclos
		0.5 ciclos
		0.5 ciclos (12 ciclos)
		2i * 1c/i = 2 ciclos
		4 i * 2 c/i = 8 ciclos
		0.5 ciclos

	2 STORE (par, impar)	2 instr. * 1 ciclo/instr. = 2 ciclos

(c) ¿Cuántas bloques de memoria ocupa el vector x?¿Qué número de ciclos de reloj supondría la ejecución de una iteración del bucle si nunca hay acierto de cache cuando se accede a un nuevo bloque de x? Razone/explique su respuesta.

El número de bloques de cache que ocupa el vector, suponiendo que comienza en un múltiplo de una línea de cache es de:

$$\frac{1022\ comp.\ \times 2^2\ B/comp.}{2^5\ B/LC} = \frac{1022}{2^3\ LC} LC = 127.75 \Rightarrow 2^7 - 1\ líneas\ de\ cache\ completas\ y\ 1\ más\ incompleta\ (24B)$$

Si no hay acierto de cache en los accesos a todos los bloques de cache que forman parte de x, los ciclos de reloj que supone la ejecución de un bucle van a depender de si en el bucle se accede a un nuevo bloque del vector o no. Cuando se accede a un nuevo bloque de memoria la trasferencia desde la memoria principal a la cache de los dos primeros 8B del bloque y, por tanto, la trasferencia de los dos primeros componentes de x de 32b del bloque, supone 60 ciclos (ver segunda tabla más arriba). Tras esos 60 ciclos se podrán ejecutar las

trasferencias desde cache al procesador (1 ciclo cada una), las operaciones FP (9 ciclos) y el resto de operaciones del bucle (3 ciclos). La ejecución de estas instrucciones supone 12 ciclos. Los siguientes 8B (siguientes dos componentes de x) estarán en la cache trascurridos 10 ciclos desde que llegaron los anteriores; por tanto, pasados 12 ciclos estarán disponibles los componentes de x a utilizar en la siguiente iteración puesto que lo están tras 10 ciclos. Cuando se acceda a un nuevo bloque de la cache, los primeros dos componentes del bloque estarán en la cache tras 60 ciclos. La operación del bucle que usa esos componentes no se podrá ejecutar hasta pasados 60 ciclos en lugar de 12 ciclos. Se supone que se accede a un bloque detrás de otro, sin interrupción. Resumiendo:

- Si se accede a un nuevo bloque, las instrucciones del bucle a partir de los load (los accesos desde cache a procesador incluidos) no podrá ejecutarse hasta que trascurran 60 ciclos.

- Si se accede a los siguientes 8B (2 componentes de x) de un bloque se ejecutará el siguiente bucle (a partir de los load) trascurridos 12 ciclos, porque 12 son los ciclos que supone la ejecución de una iteración completa y, tras 10 ciclos después de la última trasferencia de memoria principal a cache, ya se tienen los siguientes 8B del bloque en la cache.

Código optimizado	**Instrucciones máquina**	
`par=0; impar=0;`	5 instr. inialización de registros (par, impar, i, N, c)	5 instr. * 0.5 ciclos/instr. = 2.5 ciclos

`for (i=0;i<N;i=i+2)`
 	
`par=par+c*x[i];`
`impar=impar-c*x[i+1];`
	ADD i,2
	CMP i,N
	BRCND si i>=N salto a STORE
	2 LOAD (x[i],x[i+1])
	2 MULFP, 1 ADDFP, 1SUBFP
	BR a ADD
	(12 ciclos o 60 ciclos)
		0.5 ciclos
		0.5 ciclos
		0.5 ciclos
		2 i * 1 c/i = 2 ciclos
		4 i * 2 c/i = 8 ciclos
		0.5 ciclos
		(60 o 10 ciclos supone la trasferencia de MP a cache si hay fallo)

	2 STORE (par, impar)	2 instr. * 1 ciclo/instr. = 2 ciclos

(d) En caso de producirse fallos en los acceso a todos los bloques de `x`. ¿Cuántas instrucciones se ejecutan hasta finalizar el programa una vez que está en cache disponible el último dato del vector `x` con el que se opera? ¿Qué número de ciclos de reloj tarda la ejecución de estas instrucciones? Razone/explique su respuesta

Se ejecutan las instrucciones que se han destacado en verde en la siguiente tabla:

Código optimizado	**Instrucciones máquina**	
`par=0; impar=0;`	5 instr. inialización de registros (par, impar, i, N, c)	5 instr. * 0.5 ciclos/instr. = 2.5 ciclos

`for (i=0;i<N;i=i+2)`
 	
`par=par+c*x[i];`
`impar=impar-c*x[i+1];`
	ADD i,2
	CMP i,N
	BRCND si i>=N salto a STORE
	2 LOAD (x[i],x[i+1])
	2 MULFP, 1 ADDFP, 1SUBFP
	BR a ADD
		0.5 ciclos
		0.5 ciclos
		0.5 ciclos
		2i * 1c/i = 2 ciclos
		4 i * 2 c/i = 8 ciclos
		0.5 ciclos

	2 STORE (par, impar)	2 instr. * 1 ciclo/instr. = 2 ciclos

Se ejecuta una iteración completa más los dos almacenamientos que hay fuera del bucle. Su ejecución supone **14 ciclos** porque cada almacenamiento supone 1 ciclo según se indica en el enunciado y una iteración del bucle supone 12 ciclos.

(e) Optenga los ciclos que tardaría en ejecutarse todo el código si no hay ningún fallo de cache. Optenga los MFLOPS para este caso.

T(sin fallos cache) = 2.5 ciclos (inicialización) + 1.5 ciclos (1ADD+1CMP+1BRCND) +

1022/2 iteraciones* 12 ciclos/iteración (2LOAD+4FP+1BR+1ADD+1CMP+1BRCND) +

2 ciclos (STORE) = 4 ciclos + 6132 ciclos + 2 ciclos = **6138 ciclos**

Cálculo de los MFLOPS **sin contar las operaciones de acceso a memoria**:

$$\text{MFLOPS} = \frac{4\,\text{FLO}/bucle \times \frac{1022}{2}\,\text{bucles}}{\frac{6138\,\text{ciclos}}{2 \times 10^9 ciclos/seg.} \times 10^6} = \frac{2044\,\text{FLO}}{3069\,\text{seg.} \times 10^{-3}} \cong 666.01\,\text{MFLOPS}$$

(e) Optenga los ciclos que tardaría en ejecutarse todo el código en caso de producirse fallos en los acceso a todos los bloques de `x`. Optenga los MFLOPS para este caso.

Se debe tener en cuenta que x no ocupa un número entero de bloques de cache. Ocupa 2^7 bloques completos y un bloque incompleto (le faltarían 2 componentes de 32 bits para completar el bloque)

T(con fallos cache) =

{ 2.5 ciclos (inicialización) + 1.5 ciclos (1ADD+1CMP+1BRCND) } +

{ 60 ciclos + 3 × max(10,12) } (procesamiento 1er. bloque sin los últimos 8 B) +

{ (ciclos (max(60,12) ciclos + 3*max(10,12) ciclos) × (2^7-2) fallos de cache } (proc. 2^7-2 bloques sin 8B últ.) +

{ max(60,12) ciclos + 2*max(10,12) ciclos + 12 ciclos } (procesamiento último bloque con últimos 8B) +

{ 2 ciclos (STOREs) } (fuera del bucle) =

4 ciclos + (60+36) × 128 + 2 ciclos = 4 ciclos + 12288 ciclos + 2 ciclos = **12294 ciclos**

$$\text{MFLOPS} = \frac{4\,\text{FLOP}/bucle \times \frac{1022}{2}\,\text{bucles}}{\frac{12294\,\text{ciclos}}{2 \times 10^9 ciclos/seg.} \times 10^6} = \frac{2044\,\text{FLOP}}{6147\,\text{seg.} \times 10^{-3}} \cong 332.52\,\text{MFLOPS}$$

◀◀◀◀◀◀◀◀◀◀◀◀◀◀◀◀◀◀◀

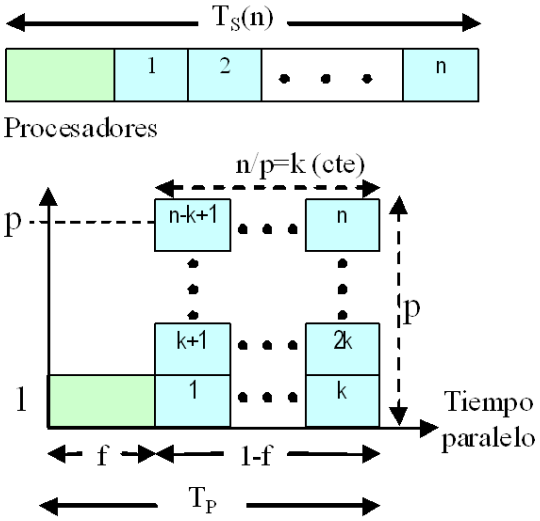
Cuestión 1. Deduzca la expresión matemática que se suele utilizar para caracterizar la ley de Gustafson. Defina claramente y sin ambigüedad el punto de partida que va a utilizar para deducir esta expresión y cada una de las etiquetas que utilice. ¿Qué nos quiere decir Gustafson con esta ley?

Solución

Punto de partida:

Se parte de un modelo de código en el que el tiempo de ejecución secuencial no permanece constante al variar el número de procesadores, lo que permanece constante es el tiempo de ejecución paralelo y en el que hay (como se representa en la figura de abajo):

1. Un trozo no paralelizable (la fracción notada por `f` en la figura)
2. Otro trozo (el resto) que se puede paralelizar repartiéndolo por igual entre los procesadores disponibles.



Las etiquetas de la figura se describen a continuación:

1. T_p es una constante que representa el tiempo de ejecución paralelo que permanece constante conforme varía p ya que k es constante
2. k es una constante igual a al grado de paralelismo del código secuencial dividido entre el número de procesadores (n/p). Cuando aumenta p aumenta también n para mantener constante k y T_p
3. $T_s(n=kp)$ es el tiempo de ejecución secuencial que varía conforme varía p
4. f es la fracción del tiempo de ejecución paralelo del código que supone la parte no paralelizable
5. $(1-f)$ es la fracción del tiempo de ejecución paralelo del código que supone la ejecución de la parte paralelizable
6. p representa el número de procesadores disponibles

Deducción:

La ganancia en prestaciones para este modelo de código ideal sería (se supone 0 el tiempo de sobrecarga):

$$S(p) = \frac{T_S(n = kp)}{T_P} = \frac{f \times T_P + (1-f) \times T_P \times p}{T_P} = f + (1-f) \times p$$

Según esta expresión la ganancia crece de forma lineal conforme varía p con una pendiente constante de $(1-f)$.

Cuestión 2. ¿Cuál de los siguientes modelos de consistencia permite mejores tiempos de ejecución? Justifique su respuesta.

a) modelo de ordenación débil
b) modelo implementado en los procesadores de la línea x86
c) modelo de consistencia secuencial
d) modelo de consistencia de liberación

Solución

(d) modelo de consistencia de liberación

Tanto el modelo de ordenación débil como el de consistencia de liberación relajan todos los órdenes entre operaciones de acceso a memoria (W->R, W->W, R->W,R) por lo que permitirán mejores tiempos de ejecución que el resto porque ningún acceso tiene que esperar a otro.
Pero el de liberación ofrece mejores prestaciones que el de ordenación débil porque:
(1) El de ordenación débil ofrece instrucciones máquina que permiten que, si S es una operación de sincronización, se garanticen los siguientes órdenes:

- S->WR
- WR->S

Con estos órdenes los accesos a memoria que hay detrás de una sincronización no pueden empezar hasta que no hayan acabado todos los accesos que hay delante de la sincronización.

(2) Mientras que el modelo de liberación ofrece instrucciones máquina menos restrictivas que permiten garantizar, si SA es una operación de adquisición y SL de liberación, los siguientes órdenes:

- SA->WR
- WR->SL

Con estos órdenes los accesos a memoria que hay detrás de una sincronización de liberación pueden empezar aunque no hayan terminado los que hay delante, y los que hay delante de una operación de adquisición pueden terminar después de las que hay detrás.