

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

Licencia Creative Commons 

### 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
<code>par=0; impar=0;</code>	5 instr. inicialización de registros ( <code>par</code> , <code>impar</code> , <code>i</code> , <code>N</code> , <code>c</code> )
<code>for (i=0;i&lt;N;i=i+2)</code>	ADD i,2 CMP i,N BRCND si i>=N salto a STORE
<code>par=par+c*x[i];</code> <code>impar=impar-c*x[i+1];</code>	2 LOAD ( <code>x[i]</code> , <code>x[i+1]</code> ) 2 MULFP, 1 ADDFP, 1 SUBFP BR a ADD
	2 STORE ( <code>par</code> , <code>impar</code> )

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
<code>par=0; impar=0;</code>	5 instr. inicialización de registros ( <code>par</code> , <code>impar</code> , <code>i</code> , <code>N</code> , <code>c</code> )
<code>for (i=0;i&lt;N;i=i+2)</code>	ADD i,2 CMP i,N BRCND si i>=N salto a STORE
<code>par=par+c*x[i];</code> <code>impar=impar-c*x[i+1];</code>	2 LOAD (x[i],x[i+1]) 2 MULFP, 1 ADDFP, 1SUBFP BR a ADD
	2 STORE ( <code>par</code> , <code>impar</code> )

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/explice su respuesta



## Solución

<b>LOAD</b>	1 instr./ciclo	1 si NO hay fallo de cache
<b>FP</b>	0.5 instr./ciclo	2

<b>Burst (ráfaga)</b>	6    –    1    –    1    –    1	Frecuencia del bus = 200 MHz => ciclo de 1/200 MHz = 5 ns => 6 ciclos = 30ns
<b>Bytes transferidos</b>	8B    8B    8B    8B = 32B	Línea de cache de 32B: se supone entonces que un <i>burst</i> obtiene una línea
<b>Ciclos componente entre MP y cache</b>	60    70    80    90 cicl... x[0]   x[2]   x[4]   x[6] x[1]   x[3]   x[5]   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.

Procesador

Cache

Memoria  
Principal

<code>par=0; impar=0;</code>	5 instr. inialización de registros (par, impar, i, N, c)	5 instr. * 0.5 ciclos/instr. = 2.5 ciclos
	ADD i,2 CMP i,N BRCND si i>=N salto a STORE	0.5 ciclos 0.5 ciclos 0.5 ciclos ✓ 1.5 ciclos
	2 STORE (par, impar)	2 instr. * 1 ciclo/instr. = 2 ciclos

<code>par=0; impar=0;</code>	5 instr. inialización de registros (par, impar, i, N, c)	5 instr. * 0.5 ciclos/instr. = 2.5 ciclos
	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 2i * 1c/i = 2 ciclos 4i * 2 c/i = 8 ciclos 0.5 ciclos
	2 STORE (par, impar)	2 instr. * 1 ciclo/instr. = 2 ciclos



<code>par=0; impar=0;</code>	5 instr. inialización de registros (par, impar, i, N, c)	5 instr. * 0.5 ciclos/instr. = 2.5 ciclos
	<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> 12 ciclos o 60 ciclos </div> <div> 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 </div> <div style="margin-left: 20px;"> 60 o 10 ciclos  --- supone la  transferencia de MP  a cache si hay fallo </div> </div>	
	2 STORE (par, impar)	2 instr. * 1 ciclo/instr. = 2 ciclos

<code>par=0; impar=0;</code>	5 instr. inialización de registros (par, impar, i, N, c)	5 instr. * 0.5 ciclos/instr. = 2.5 ciclos
	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 2 i * 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



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

$$T(\text{sin fallos cache}) = 2.5 \text{ ciclos (inicialización)} + 1.5 \text{ ciclos (1ADD+1CMP+1BRCND)} + \\ 1022/2 \text{ iteraciones} * 12 \text{ ciclos/iteración (2LOAD+4FP+1BR+1ADD+1CMP+1BRCND)} + \\ 2 \text{ ciclos (STORE)} = 4 \text{ ciclos} + 6132 \text{ ciclos} + 2 \text{ ciclos} = \mathbf{6138 \text{ ciclos}}$$

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

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

(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) =

$$\{ 2.5 \text{ ciclos (inicialización)} + 1.5 \text{ ciclos (1ADD+1CMP+1BRCND)} \} + \\ \{ 60 \text{ ciclos} + 3 \times \max(10,12) \} \text{ (procesamiento 1er. bloque sin los últimos 8 B)} + \\ \{ (\text{ciclos}(\max(60,12) \text{ ciclos} + 3 * \max(10,12) \text{ ciclos}) \times (2^7 - 2) \text{ fallos de cache} \} \text{ (proc. } 2^7 - 2 \text{ bloques sin 8B últ.)} + \\ \{ \max(60,12) \text{ ciclos} + 2 * \max(10,12) \text{ ciclos} + 12 \text{ ciclos} \} \text{ (procesamiento último bloque con últimos 8B)} + \\ \{ 2 \text{ ciclos (STOREs)} \} \text{ (fuera del bucle)} = \\ 4 \text{ ciclos} + (60+36) \times 128 + 2 \text{ ciclos} = 4 \text{ ciclos} + 12288 \text{ ciclos} + 2 \text{ ciclos} = \mathbf{12294 \text{ ciclos}}$$

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



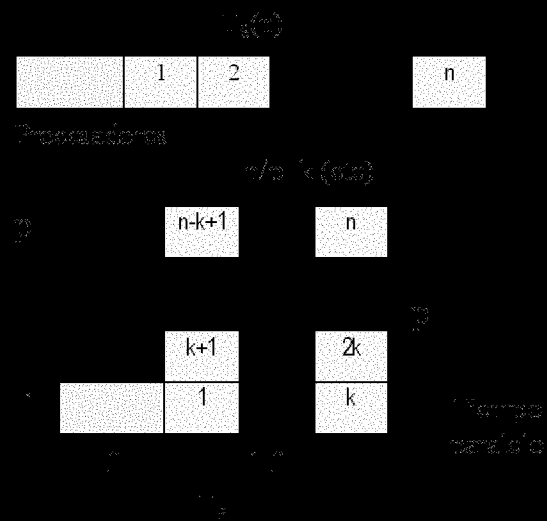
**Cuestión 1.** Deduzca la expresión matemática que se suele utilizar para caracterizar la ley de Gustafson. Defina claramente y sin ambigüedad el punto de partida que va a utilizar para deducir esta expresión y cada una de las etiquetas que utilice. ¿Qué nos quiere decir Gustafson con esta ley?

**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  $\epsilon$  en la figura)
2. Otro trozo (el resto) que se puede paralelizar repartíéndolo por igual entre los procesadores disponibles.



1.

2.

3.

4.

5.

6.



Cuestión 2.

Solución





Tanto el modelo de ordenación débil como el de consistencia de liberación relajan todos los órdenes entre operaciones de acceso a memoria ( $W \rightarrow R$ ,  $W \rightarrow W$ ,  $R \rightarrow W, R$ ) 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:

- $S \rightarrow WR$
- $WR \rightarrow S$

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:

- $SA \rightarrow WR$
- $WR \rightarrow SL$

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.

