Universidad Carlos III de Madrid

Departamento de Informática
Grado en Ingeniería Informática
Arquitectura de Computadores

Examen

ARCOS

ATENCIÓN:

- Lea atentamente todo el enunciado antes de comenzar a contestar.
- **La duración del examen es de 180 minutos.**

NOMBRE:
APELLIDOS:
NIA:

Ejercicio 1: (1,5 puntos)

Se dispone de un computador con un solo núcleo que ejecuta una aplicación de evaluación de riesgos financieros. Esta aplicación es intensiva en cálculo, a lo que dedica el 90% del tiempo. El 10% restante lo dedica a esperar en operaciones de entrada/salida a disco.

Del tiempo que la aplicación pasa ejecutando instrucciones de cálculo un 75% del tiempo lo pasa ejecutando operaciones en coma flotante y un 25% lo pasa ejecutando otras instrucciones. La ejecución de una instrucción de coma flotante requiere como promedio 12 CPI. El resto de instrucciones requieren como promedio 4 CPI.

Se está valorando la migración de esta aplicación a las siguientes alternativas, que no incorporan ninguna mejora para el tiempo de las operaciones de entrada/salida a disco:

- Alternativa A: Un procesador con un solo núcleo y con una frecuencia de reloj un 50% más alta que la de la máquina original en el que las instrucciones de coma flotante requieren un 10% más de ciclos por instrucción y el resto de instrucciones requieren un 25% más de ciclos por instrucción.
- Alternativa B: Un procesador con cuatro núcleos y con una frecuencia de reloj un 50% más baja que la de la máquina original, en el que las instrucciones de coma flotante requieren un 20% menos de ciclos de reloj y el resto de instrucciones los mismos ciclos de reloj.

Se pide responder de forma justificada a las siguientes cuestiones:

a) ¿Cuál será la aceleración/deceleración global de la aplicación en el caso A?
b) ¿Cuál será la aceleración/deceleración global de la aplicación en el caso B si se asume que la parte de cálculo es totalmente paralelizable mientras la entrada/salida no admite ningún tipo de paralelización?

Solución:

El tiempo dedicado a la ejecución de instrucciones en el computador original será:

$T(inst,orig) = 0.75 * 12 * IC * P + 0.25 * 4 * IC * P = (9+1) * IC * P$

Alternativa A:

El tiempo dedicado a la ejecución de instrucciones en el computador A será:

T(inst,A) =(0.75 * (1,1 * 12) + 0.25 * (1.25 * 4)) * IC * (P/1.5) = (9.9+1.25) * IC * P / 1.5 = 11,15 / 1.5* IC * P = 7.433 * IC *P

El Speedup debido a instrucciones será:

S(inst,A) = T(inst,orig) / T(inst,A) = 10 / 7.433 = 1.345

Aplicando la Ley de Amdahl el speedup global sera:

S(A) = 1 / (0.1 + 0.9 / 1.345) = 1.3

Alternativa B:

En este caso, al asumirse paralelización completa de la parte de cálculo se puede considerar que el número de instrucciones a ejecutar en cada núcleo es la cuarta parte del original.

T(inst,B) = (0.75 * 0.8 * 12 + 0.25 *4) * (IC / 4) * (P /0.5) = (7.2+1) * 2 / 4 * IC * P = 4.1 * IC * P

El Speedup debido a instrucciones sera:

S(inst,B) = T(inst,orig) / T(inst,B) = 10/4.1 = 2,439

Aplicando la Ley de Amdahl el speedup global sera:

S(B) = 1 / (0.1 + 0.9 / 2,439) = 2,132

Ejercicio 2: (1,5 puntos)

Responda a las siguientes cuestiones:

a) Diga cuándo un sistema de memoria compartida es coherente (máximo 3-4 líneas).
b) Enumere las condiciones para la coherencia de memoria.
c) Enumere las restricciones que garantizan que un sistema ofrece consistencia secuencial.
d) En un modelo de consistencia de adquisición/liberación ¿Cuál es la semántica de la operación de adquisición (*acquire*)?
e) En un modelo de consistencia de adquisición/liberación ¿Cuál es la semántica de la operación de liberación (*release*)?

Solución:

Apartado A:

Un sistema de memoria es coherente si cualquier lectura de una posición devuelve el valor más reciente que se haya escrito para esa posición.

Apartado B:

1. Preservación de orden de programa.
2. Vista coherente de la memoria.
3. Serialización de las escrituras.

Apartado C:

1. Orden de programa
2. Atomicidad.

Apartado D:

La operación de adquisición debe completarse antes que todos los accesos a memoria subsiguientes.

Apartado E:

- La operación de liberación debe completarse antes que todos los accesos a memoria previos.
- Si se pueden iniciar accesos a memoria posteriores.
- Las operaciones que siguen a una liberación y que deban esperar se deben proteger con una adquisición.

Ejercicio 3: (1 punto):

Explique brevemente las diferencias que hay en OpenMP entre planificación estática, planificación dinámica y planificación guiada.

Solución:

La planificación define como se asignan las iteraciones de un bucle a un conjunto de hilos.

Planificación estática:

Todos los hilos reciben la asignación de iteraciones antes comenzar la ejecución del bucle. Por defecto cada hilo recibe un número igual de iteraciones.

Planificación dinámica:

Se dividen las iteraciones en conjuntos de un cierto tamaño (*chunk size*). Cuando un hilo termina de ejecutar un conjunto de iteraciones se le asigna dinámicamente otro de estos conjuntos.

Planificación guiada:

Es similar a la planificación dinámica, pero el tamaño del conjunto asignado se va reduciendo con cada asignación.

Ejercicio 4: (2 puntos)

En un determinado procesador se pretende ejecutar el siguiente segmento de código:

```
i0: lw $r4, 0($r1)
i1: lw $r5, 0($r2)
i2: add $r4, $r4, $r5
i3: sw $r4, 0($r3)
i4: addi $r1, $r1, 4
i5: addi $r2, $r2, 4
i6: addi $r3, $r3, 4
i7: bne $r3, $r0, i0
```

Asuma que el procesador tiene una arquitectura segmentada de 5 etapas (captación, decodificación, ejecución, memoria y post-escritura) sin envío adelantado (*forwarding*). Todas las operaciones se ejecutan en un ciclo por etapa, excepto:

- Las operaciones de carga y almacenamiento, que requieren un total de dos ciclos para la etapa de memoria (un ciclo adicional).
- Las instrucciones de salto, que requieren un ciclo adicional en la etapa de ejecución. Considere que estas instrucciones no cuentan con ningún tipo de predicción de saltos.

Se pide responder de forma justificada a las siguientes cuestiones:

a) Determine los riesgos de datos RAW que presenta el código que tienen impacto en la ejecución del código.
b) Muestre un diagrama de tiempos con las fases de ejecución de cada instrucción para una iteración.
c) Determine cuantos ciclos requiere la ejecución de una iteración del bucle si no hay ningún tipo de predicción de saltos.
d) Proponga un desenrollamiento (*loop unrolling*) del bucle asumiendo que bucle se ejecuta 1000 iteraciones. Desenrolle con un factor de cuatro iteraciones.
e) Determine la aceleración o *speedup* obtenido mediante el desenrollamiento del apartado anterior.

Solución:

Apartado A:

$r4: i0 -> i2
$r5: i1 -> i2
$r4: i2 -> i3
$r3: i6 -> i7

Apartado B:

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
I0	F	D	X	M1	M2	W																
I1		F	D	X	-	M1	M2	W														
I2			F	-	-	-	-	D	X	M	W											
I3								F	-	-	D	X	M1	M2	W							
I4											F	D	X	-	M	W						
I5												F	D	-	X	M	W					
I6													F	-	D	X	M	W				
I7															F	-	-	D	X1	X2	M	W

- I0: Requiere dos ciclos de memoria.
- I1: No puede empezar etapa de memoria hasta que I0 no termina etapa de memoria. Requiere dos ciclos de memoria.
- I2: No puede empezar a decodificar hasta que I1 no hace WB.
- I3: No puede empezar captación hasta que se libera unidad por I2.
- I4: No puede empezar etapa de memoria hasta que I3 libera unidad de memoria.
- I5: No puede empezar etapa de ejecución hasta que I4 libera unidad de ejecución.
- I6: No puede empezar etapa de decodificación hasta que I5 libera unidad de decodificación.
- I7: No puede empezar decodificación hasta que I6 hace WB.

Apartado C:

Si se considera que la etapa de decodificación incluye un comparador sobre el banco de registros la siguiente instrucción a I7 puede comenzar después de que finalice la etapa de decodificación de I7 (esto es en el ciclo 19) por lo que cada iteración requiere 18 ciclos de reloj.

Apartado C (solución alternativa):

Si no se considera que la etapa de decodificación incluye un comparador, la comparación de dos registros debe hacerse con la ALU general y por tanto, no se podrá tomar la decisión hasta que haya finalizado la etapa de ejecución de I7 (en el ciclo 21). En este caso cada iteración requiere 20 ciclos de reloj.

Apartado D:

A continuación se presenta una posible solución. Obsérvese, sin embargo, que son posibles soluciones más agresivas que podrían dar lugar a mejores *speedups*.

```
I0: lw $r4, 0($r1)
i1: lw $r5, 0($r2)
i2: lw $r6, 4($r1)
i3: lw $r7, 4($r2)
i4: lw $r8, 8($r1)
i5: lw $r9, 8($r2)
i6: lw $r10, 12($r1)
```

```
i7: lw $r11, 12($r2)
i8: add $r4, $r4, $r5
i9: add $r6, $r6, $r7
i10: add $r8, $r8, $r9
i11: add $r10, $r10, $r11
i12: sw $r4, 0($r3)
i13: sw $r6, 4($r3)
i14: sw $r8, 8($r3)
i15: sw $r10, 12($r3)
i16: addi $r3, $r3, 16
i17: addi $r2, $r2, 16
i18: addi $r1, $r1, 16
i19: bne $r3, $r0, i0
```

Apartado E:

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I0	F	D	X	M1	M2	W														
I1		F	D	X	-	M1	M2	W												
I2			F	D	-	X	-	M1	M2	W										
I3				F	-	D	-	X	-	M1	M2	W								
I4						F	-	D	-	X	-	M1	M2	W						
I5								F	-	D	-	X	-	M1	M2	W				
I6										F	-	D	-	X	-	M1	M2	W		
I7												F	-	D	-	X	-	M1	M2	W
I8														F	-	D	-	X	-	M
I9																F	-	D	-	X
I10																		F	-	D
I11																				F

Instr.	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
I8	W																			
I9	M	W																		
I10	X	M	W																	
I11	D	X	M	W																
I12	F	D	X	M1	M2	W														
I13		F	D	X	-	M1	M2	W												
I14			F	D	-	X	-	M1	M2	W										
I15				F	-	D	-	X	-	M1	M2	W								
I16						F	-	D	-	X	-	M	W							
I17								F	-	D	-	X	M	W						
I18										F	-	D	X	M	W					
I19												F	D	X	X	M	W			

Dependiendo del criterio elegido en el apartado B, el número de ciclos para iteración desenrrollada será 33 o 35 y el número de ciclos por iteración será 33/4 = 8.25 o 35/4 = 8.75

Por tanto el speedup será:

S = 18 / 8.25 = 2.18

O bien:

S = 20/ 8.75 = 2.28

Ejercicio 5: (2 puntos)

Sea un multiprocesador con arquitectura de memoria compartida simétrica basado en bus con protocolo de espionaje o *snooping*. Cada procesador tiene una caché privada cuya coherencia se mantiene usando el protocolo MSI. Cada caché utiliza correspondencia directa y tiene cuatro bloques cada uno con dos palabras. Esta caché utiliza como campo de etiqueta la dirección de memoria completa.

Las siguientes tablas muestran el estado de cada memoria, con la palabra menos significativa a la izquierda.

Procesador P0

Bloque	Estado	Etiqueta	Datos	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	S	0x00100708	0x00000000	0x00001234
B2	M	0x00100710	0x00000000	0x0077AABB
B3	I	0x00100718	0x00000000	0x7FAABB11

Procesador P1

Bloque	Estado	Etiqueta	Datos	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	M	0x00100728	0x00000000	0xFF000000
B2	I	0x00100710	0x00000000	0xEEEE7777
B3	S	0x00100718	0x00000000	0x7FAABB11

Procesador P2

Bloque	Estado	Etiqueta	Datos	
B0	S	0x00100720	0x00000000	0x1111AAAA
B1	S	0x00100708	0x00000000	0X00001234
B2	I	0x00100710	0x00000000	0x7FAABB11
B3	I	0x00100718	0x00001234	0x1111AABB

Para cada uno de los apartados que a continuación se presentan, parta de la situación inicial del problema, sin tener en cuenta los cambios de los apartados anteriores. Indique los cambios que se producen en las cachés. En el caso de las lecturas, indique además cuál es el valor efectivamente leído.

a) P2: write 0x00100708, 0xFFFFFFFF
b) P2: read 0x00100708
c) P2: read 0x00100718

En cada apartado deberá rellenar una tabla con el siguiente formato, justificando la respuesta:

Procesador	Bloque	Estado	Estado	Etiqueta	Datos

		Anterior	Nuevo			

Solución:

Apartado A:

Se produce una escritura en el bloque B1 de la caché de P2 que se encuentra en estado S. Esto produce los siguientes cambios en P2:

- Se pasa a estado exclusivo (M).
- Se coloca una invalidación en el bus.

La invalidación es ignorada por el procesador P1, pero es tratada por el procesador P0.

En P0 se producen los siguientes cambios:

- Se pasa el estado a inválido (I)

En este caso no se producen modificaciones en la memoria principal.

Procesador P0

Bloque	Estado	Etiqueta	Datos	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	I	0x00100708	0x00000000	0x00001234
B2	M	0x00100710	0x00000000	0x0077AABB
B3	I	0x00100718	0x00000000	0x7FAABB11

Procesador P1

Bloque	Estado	Etiqueta	Datos	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	M	0x00100728	0x00000000	0xFF000000
B2	I	0x00100710	0x00000000	0xEEEE7777
B3	S	0x00100718	0x00000000	0x7FAABB11

Procesador P2

Bloque	Estado	Etiqueta	Datos	
B0	S	0x00100720	0x00000000	0x1111AAAA

B1	M	0x00100708	0xFFFFFFFF	0X00001234
B2	I	0x00100710	0x00000000	0x7FAABB11
B3	I	0x00100718	0x00001234	0x1111AABB

Tabla de cambios

Procesador	Bloque	Estado Anterior	Estado Nuevo	Etiqueta	Datos	
P2	B1	I	M	0X00100708	0XFFFFFFFF	0X00001234
P0	B1	S	I	0X00100708	0X00000000	0X00001234

Apartado B:

Se produce una lectura en el bloque B1 de la caché P2 que se encuentra en el estado S. Se trata de un acierto y no se produce ningún cambio en las cachés o en memoria.

El valor leído es: 0x00000000

Apartado C:

Se produce un fallo de lectura la caché de P2 para el bloque B3 que está en estado inválido (I). Por tanto se coloca el fallo de lectura en el bus y se pasa a estado compartido (S).

El procesador P0, tiene este bloque en estado inválido e ignora el fallo de lectura.

El procesador P1, tiene este bloque en estado compartido (S) y continúa en este estado.

El valor leído es 0x00000000

Procesador P0

Bloque	Estado	Etiqueta	Datos	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	S	0x00100708	0x00000000	0x00001234
B2	M	0x00100710	0x00000000	0x0077AABB
B3	I	0x00100718	0x00000000	0x7FAABB11

Procesador P1

Bloque	Estado	Etiqueta	Datos	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	M	0x00100728	0x00000000	0xFF000000
B2	I	0x00100710	0x00000000	0xEEEE7777
B3	S	0x00100718	0x00000000	0x7FAABB11

Procesador P2

Bloque	Estado	Etiqueta	Datos	
B0	S	0x00100720	0x00000000	0x1111AAAA
B1	S	0x00100708	0x00000000	0X00001234
B2	I	0x00100710	0x00000000	0x7FAABB11
B3	**S**	0x00100718	**0x00000000**	**0x7FAABB11**

Tabla de cambios

Procesador	Bloque	Estado Anterior	Estado Nuevo	Etiqueta	Datos	
P2	B3	I	S	0X00100718	0X00000000	0X7FAABB11

Ejercicio 6: (2 puntos)

Sea la siguiente función:

```
std::mutex m; // mutex global
int contador; // contador global

void f() {
  m.lock();
  ++contador;
  m.unlock();
}
```

Se desea sustituir la variable global **m** y evitar posibles llamadas al sistema operativo, pero al mismo tiempo se desea garantizar la exclusión mutua en el incremento del contador.

Se pide:

a) Proponga y codifique una solución que ofrezca consistencia secuencial y no implique llamadas al sistema.
b) Proponga y codifique una solución que ofrezca consistencia de adquisición-liberación.
c) Proponga y codifique una solución que ofrezca consistencia de adquisición-liberación y sea válida en el caso en que el contador pase a ser una variable de tipo **double**.

Solución:

Apartado A:

```
std::atomic<int> contador; // contador global

void f() {
  ++contador;
}
```

Apartado B:

```
std::atomic<int> contador; // contador global

void f() {
  contador.fetch_add(1, std::memory_order_acq_rel);
}
```

Apartado C:

```
std::atomic_flag spin = ATOMIC_FLAG_INIT;

double contador; // contador global
```

```
void f() {
  while (spin.test_and_set(std::memory_order_acquire)) {}
  contador += 1.0;
  spin.clear(std::memory_order_release);
}
```