Jiuxe
www.wuolah.com/student/Jiuxe

3434

AC - Relacion 3 Algunos Ejercicios Resueltos.pdf

Ejercicios Resueltos

2° Arquitectura de Computadores

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
UGR - Universidad de Granada

2º curso / 2º cuatr.

Grado en Ing. Informática

Arquitectura de Computadores. Algunos ejercicios resueltos

Tema 3. Arquitecturas con paralelismo a nivel de thread (TLP)

Profesores responsables: Mancia Anguita, Julio Ortega

1 Ejercicios

Ejercicio 1. En un multiprocesador SMP con 4 procesadores o nodos (N0-N3) basado en un bus, que implementa el protocolo MESI para mantener la coherencia, supongamos una dirección de memoria incluida en un bloque que no se encuentra en ninguna cache. Indique los estados de este bloque en las caches y las acciones que se producen en el sistema ante la siguiente secuencia de eventos para dicha dirección:

1. Lectura generada por el procesador 1 2. Lectura generada por el procesador 2
3. Escritura generada por el procesador 1
4. Escritura generada por el procesador 2
5. Escritura generada por el procesador 3

Solución

Datos del ejercicio

Se accede a una dirección de memoria cuyo bloque `k` no se encuentra en ninguna cache, luego debe estar actualizado en memoria principal y el estado en las caches se considera inválido.

Estado del bloque en las caches y acciones generadas ante los eventos que se refiere a dicho bloque

Hay 4 nodos con cache y procesador (N0-N3). Intervienen N1, N2 y N3. En la tabla se van a utilizar las siguentes siglas y acrónimos:

MP: Memoria Principal.

PtLec(k): paquete de petición de lectura del bloque k.

PtLecEx(k): paquete de petición de lectura del bloque k y de petición de acceso exclusivo al bloque k.

RpBloque(k): paquete de respuesta con el bloque k.

Se va a suponer que no existe en el sistema paquete de petición de acceso exclusivo a un bloque sin lectura (no existe PtEx).

ESTADO INICIAL	EVENTO	ACCIÓN	ESTADO SIGUIENTE

N1) Inválido **N2) Inválido** **N3) Inválido**	P1 lee k	1.- N1 (el controlador de cache de N1) genera y deposita en el bus una petición de lectura del bloque k (PtLec(k)) porque no lo tiene en su caché válido 2.-MP (el controlador de memoria de MP), al observar PtLec(k) en el bus, genera la respuesta con el bloque (RpBloque(k)).	**N1) Exclusivo** **N2) Inválido** **N3) Inválido**
		3.- N1 (el controlador de cache de N1) recoge del bus la respuesta depositada por la memoria principal (RpBloque(k)), el bloque entra en la cache de N1 en estado exclusivo ya que no hay copia en otra cache del bloque (es decir, la salida de la OR cableada con entradas procedentes de todas las caches es 0).	
N1) Exclusivo **N2) Inválido** **N3) Inválido**	P2 lee k	1.- N2 genera y deposita en el bus una PtLec(k) porque no tiene k en su caché en estado válido 2.- N1 observa PtLec(k) en el bus y, como tiene el bloque en estado exclusivo, lo pasa a compartido (la copia que tiene ya no es la única válida en caches). MP, al observar PtLec(k) en el bus, genera la respuesta con el bloque (RpBloque(k)). 3.- N2 recoge RpBloque(k) que ha depositado la memoria, el bloque entra en estado compartido en la cache de N2 (la salida de la OR cableada será 1).	**N1) Compartido** **N2) Compartido** **N3) Inválido**
N1) Compartido **N2) Compartido** **N3) Inválido**	P1 escribe en k	1.- N1 genera petición de lectura con acceso exclusivo del bloque k (PtLecEx(k)) (suponemos que no hay petición de acceso exclusivo sin lectura, no hay PtEx). N1 modifica la copia de k que tiene en su cache y lo pasa a estado modificado. 2.- N2 observa PtLecEx(k) y, como la petición incluye acceso exclusivo (Ex) a un bloque que tiene en su cache en estado compartido, pasa su copia a estado inválido. MP genera RpBloque(k) porque observa en el bus una petición de k con lectura (Lec), pero esta respuesta no se va a recoger del bus. N1 no recoge RpBloque(k) depositada por la memoria porque tiene el bloque válido.	**N1) Modificado** **N2) Inválido** **N3) Inválido**
N1) Modificado **N2) Inválido** **N3) Inválido**	P2 escribe en k	1.- N2 genera petición de lectura con acceso exclusivo de k (PtLecEx(k)) 2.- N1 observa PtLecEx(k) y, como tiene el bloque en estado modificado (es la única copia válida en todo el sistema), inhibe la respuesta de MP y genera respuesta con el bloque RpBloque (k), y además, como el paquete pide acceso exclusivo a k (Ex), invalida su copia del bloque k. 3.- N2 recoge RpBloque(k), introduce k en su cache, lo modifica y lo pone en estado modificado	**N1) Inválido** **N2) Modificado** **N3) Inválido**
N1) Inválido **N2) Modificado** **N3) Inválido**	P3 escribe en k	1.- N3 genera petición de lectura con acceso exclusivo de k PtLecEx(k) 2.- N2 observa PtLecEx(k) y, como tiene el bloque en estado modificado, inhibe la respuesta de MP y genera respuesta con el bloque RpBloque (k), y además, como el paquete pide acceso exclusivo a k (Ex), invalida su copia de k. 3.- N3 recoge RpBloque(k), introduce el k en su cache, lo modifica y lo pone en estado modificado	**N1) Inválido** **N2) Inválido** **N3) Modificado**

Ejercicio 2. .

Ejercicio 3. .

Ejercicio 4. Supongamos que se va a ejecutar en paralelo el siguiente código (inicialmente *x* e *y* son 0):

P1	**P2**

`x=1; x=2;`
`print y ;`
	`y=1;`
	`y=2;`
	`print x`
	`;`

Qué resultados se pueden imprimir si (considere que el compilador no altera el código):

(a) Se ejecutan P1 y P2 en un multiprocesador con consistencia secuencial.

(b) Se ejecutan en un multiprocesador basado en un bus que garantiza todos los órdenes excepto el orden W→R. Esto es debido a que los procesadores tienen buffer de escritura, permitiendo el procesador que las lecturas en el código que ejecuta adelanten a las escrituras que tiene su buffer. Obsérvese que hay varios posibles resultados.

Solución

El compilador no altera ningún orden garantizado ya que se supone, según el enunciado, que no altera el código.

(a) Si P1 es el primero que imprime puede imprimir 0, 1 o 2, pero P2 podrá imprimir sólo 2. Esto es así porque se mantiene orden secuencial (el hardware parece ejecutar los accesos a memoria del código que ejecuta un procesador en

P1	**P2**

`(1.1) x=1;`
`(1.2) x=2;`
`(1.3) print y ;`
	`(2.1) y=1;`
	`(2.2) y=2;`
	`(2.3) print x ;`

el orden en el que están en dicho código) y, por tanto, cuando P1 lee "y" (instrucción 1.3 en el código, esta instrucción lee "y" para imprimir su contenido), ha asignado ya a "x" un 2 (punto 1.2 en el código) ya que esta asignación está antes en el código que la lectura de "y".

Si P2 es el primero que imprime podrá imprimir 0, 1 o 2, pero entonces P1 sólo puede imprimir 2. Esto es así porque se mantiene orden secuencial y, por tanto, cuando P2 lee "x" (punto 2.3 en el código), ha asignado ya a "y" un 2 (punto 2.2 en el código) ya que esta asignación está antes en el código que la lectura de "x" y se mantiene orden secuencial en los accesos a memoria, es decir, los accesos parecen completarse en el orden en el que se encuentran en el código.

Se puede obtener como resultado de la ejecución las combinaciones que hay en cada una de las líneas:

P1 P2
0 2 (en este caso P1 imprime 0 y P2 imprime 2)
1 2
2 2
2 0
2 1

(b) Si no se mantiene el orden W→R además de los resultados anteriores, los dos procesos pueden imprimir:

P1 P2
1 1 (en este caso P1 imprime 1 y P2 imprime 1)
0 1
0 2
1 0
2 0
0 0

Se pueden imprimir también las combinaciones anteriores porque no se asegura que cuando un procesador ejecute la lectura de la variable que imprime `print` (puntos 1.3 y 2.3 en los códigos) haya ejecutado las instrucciones anteriores que escriben en x (P1 en los puntos 1.1 y 1.2 del código) o en y (P2 en los puntos 2.1 y 2.2). Esto es así porque no se garantiza el orden W->R y, por tanto, una lectura puede adelantar a escrituras que estén antes en el código secuencial. P1 puede leer y (1.3) antes de escribir en x 2 (1.2) o incluso antes de escribir en x 1 (1.1). Igualmente P2 puede leer x (2.3) antes de escribir en y 2 (2.2) o antes de escribir en y 1 (2.1).

Teniendo esto en cuenta P1 puede imprimir 1 o 2 o 0, y P2 1 o 2 o 0. Todas las combinaciones son posibles.

◀◀◀◀◀◀

Ejercicio 5.

◀

Ejercicio 6. .

◀

Ejercicio 7. .

◀

Ejercicio 8. .

◀

Ejercicio 9. .

Ejercicio 10. Se quiere paralelizar el siguiente ciclo de forma que la asignación de iteraciones a los procesadores disponibles se realice en tiempo de ejecución (dinámicamente):

```
For (i=0; i<100; i++) {
     Código que usa i
                    }
```

Nota: Considerar que las iteraciones del ciclo son independientes, que el único orden no garantizado por el sistema de memoria es W->R, que las primitivas atómicas garantizan que sus accesos a memoria se realizan antes que los accesos posteriores y que el compilador no altera el código.

(a) Paralelizar el ciclo para su ejecución en un multiprocesador que implementa la primitiva `Fetch&Or` para garantizar exclusión mutua.

(b) Paralelizar el anterior ciclo en un multiprocesador que además tiene la primitiva `Fetch&Add`.

Solución

Se debe tener en cuenta que el único orden que no garantiza el hardware es el orden W->R.

(a) Paralelizar el ciclo para su ejecución en un multiprocesador que implementa la primitiva `Fetch&Or` para garantizar exclusión mutua.

Bucle

```
for (i=0;i<100;i++) {
    código para i   }
```

Dinámica

```
while (Fetch_&_Or(k,1)==1) {};
//lock(k)    n=i;  i=i+1;
```

```
mov k,0;  //unlock(k) while (n<100) {    código para n ;
while (Fetch_&_Or(k,1)==1) {};
    n=i;
i=i+1;      mov
k,0;  //unlock(k)
```

Nota: la variable i estaría inicializada a 0 (por ejemplo, se puede iniciar cuando se declara)

Se supone que el compilador no cambia de sitio "`mov k,0`".

(b) Paralelizar el anterior ciclo en un multiprocesador que además tiene la primitiva `Fetch&Add`.

Bucle

```
{
for (i=0;i<10
    código
}
```

Dinámica

```
n=Fetch_&_Add(i,1
(n<100)          cód
                 {
                    ;
}                _&_Add(i,1
```

Nota: la variable i estaría inicializada a 0 (por ejemplo, se puede iniciar cuando se declara)

Ejercicio 11.

Ejercicio 12.

Ejercicio 13. Se ha extraído la siguiente implementación de cerrojo (spin-lock) para x86 del kernel de Linux (http://lxr.free-electrons.com/source/arch/x86/include/asm/spinlock.h):

```
typedef struct {

unsigned int slock;  }

raw_spinlock_t;

…

/*Para un número de procesadores menor que 256=2^8
-#if (NR_CPUS < 256)
…
-static __always_inline void __ticket_spin_lock(raw_spinlock_t *lock)
-{
-           short inc = 0x0100;
-
-           asm volatile (
-           "lock  xaddw %w0, %1\n"  /*w: se queda con los 16 bits menos significativos*/
-      "1:                \t"    /*b: se queda con el byte menos significativo*/
-      "cmpb %h0, %b0     \n\t"  /*h: coge el byte que sigue al menos significativo*/
-      "je 2f             \n\t"  /*f: forward */
-      "rep ; nop         \n\t"  /*retardo, es equivalente a pause*/
-      "movb %1, %b0      \n\t"
-           /* don't need lfence here, because loads are in-order */
-           "jmp 1b        \n"    /*b: backward */
-           "2:"
-           : "+Q" (inc), "+m" (lock->slock) /*%0 es inc, %1 es lock->slock */
-           /*Q asigna cualquier registro al que se pueda acceder con rh: a, b, c y d; ej.
           ah, bh …
*/ -
:
-           : "memory", "cc");
-}
-
-static __always_inline void __ticket_spin_unlock(raw_spinlock_t *lock)
-{
-           asm volatile( "incb %0"    /*%0 es lock->slock */
-           : "+m" (lock->slock) -           :
-           : "memory", "cc");


-}
```

Conteste a las siguientes preguntas:

(a) Utiliza una implementación de cerrojo con etiquetas ¿Cuál es el contador de adquisición y cuál es el contador de liberación?

(b) Describa qué hace `xaddw %w0, %1` ¿opera con el contador de adquisición, con el de liberación o con los dos? ¿qué operaciones hace con ellos?

(c) Describa qué hace `cmpb %h0, %b0` ¿opera con el contador de adquisición, con el de liberación o con los dos? ¿qué operaciones hace con ellos?

(d) ¿Por qué cree que se usa el prefijo `lock` delante de la instrucción `xaddw`?

NOTAS: (1) Puede consultar las instrucciones en el manual de Intel con el repertorio de instrucciones (Volumen 2 o volúmenes 2A, 2B y 2C) que puede encontrar aquí http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html .

(2) Si no recuerda la interfaz entre C/C++ y ensamblador en `gcc` (se ha presentado en Estructura de Computadores), consulte el manual de `gcc` aquí http://gcc.gnu.org/onlinedocs/gcc-4.6.2/gcc/ExtendedAsm.html#Extended-Asm (http://gcc.gnu.org/onlinedocs/)

Solución

(b) `lock->slock` contiene el contador de liberación en los bits de 0 a 7 liberación (`lock->slock[7...0]`) y el de adquisición en los bits de 8 a 15 (`lock->slock[15...8]`).

(c) `xaddw %w0, %1` almacena en los 16 bits menos significativos del registro al que se ha asigna `inc` (%0) los 16 bits (sufijo w) menos significativos de `lock->slock` (%1) y asigna a `lock->slock` (contador de adquisición y contador de liberación) el resultado de sumarlo con `inc`. Como consecuencia: **(1)** incrementa en uno el contador de adquisición (`lock->slock[15...8]`) dado que `inc` tiene un 1 en el bit 8 (`inc` contiene 0x0100) y **(2)** almacena en `inc[15...8]` (%h0) el valor de este contador antes de la modificación y en `inc[7...0]` (%b0) el valor del contador de liberación (`lock->slock[7...0]`).

(d) `cmpb %h0, %b0` compara el valor actual del contador de liberación `inc[7...0]` (%b0) y el de adquisición `inc[15...8]` (%h0); es decir, resta ambos contadores modificando sólo el registro de estado. En las instrucciones posteriores se usa el resultado de la comparación (los bits de estado resultantes de la comparación). Si son iguales ambos contadores (bit z del registro de estado a 1), abandona la función `lock` del cerrojo, y si son distintos actualiza el valor del contador de liberación cargando lo que hay en `lock->slock[7...0]` en `inc[7...0]` (%b0)

(e) Se requiere el prefijo `lock` para que la lectura y escritura en memoria que realiza la instrucción `xaddw` se hagan de forma atómica. Si `xaddw` no fuese atómica dos flujos de control podrían leer el mismo valor del contador de adquisición y, como consecuencia, más de un flujo podría entrar a la vez en una sección crítica.

◀◀◀◀◀◀