2º curso / 2º cuatr.

Grado en Ing. Informática

# Arquitectura de Computadores. Ejercicios

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

Material elaborado por los profesores responsables de la asignatura:
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

**Ejercicio 2.** Para un multiprocesador de memoria distribuida con 8 nodos se quiere implementar un protocolo para mantenimiento de coherencia basado en directorios. Suponiendo que se necesitan un bit de estado para un bloque en el directorio de memoria principal y que el tamaño de una línea de cache es de 64 bytes, calcular el porcentaje del tamaño de memoria principal que supone el tamaño del directorio de vector de bits completo.

**Ejercicio 3.** Suponga que en un CC-NUMA de red estática de 4 nodos (N0-N3) se implementa un protocolo MSI basado en directorios sin difusión con dos estados en el directorio (válido e inválido). Cada nodo tiene 8 GBytes de memoria y una línea de cache supone 64 Bytes. Considere que el directorio utiliza vector de bits completo. **(a)** Calcule el tamaño del directorio de uno nodo en bytes. **(b)** Indique cual sería el contenido del directorio, las transiciones de estados (en cache y en el directorio) y la secuencia de paquetes generados por el protocolo de coherencia en los siguientes accesos sobre una dirección D que se encuentra en la memoria del nodo 3 (inicialmente D no está en ninguna cache):

1. Lectura generada por el procesador del nodo 1
2. Escritura generada por el procesador del nodo 1
3. Lectura generada por el procesador del nodo 2
4. Lectura generada por el procesador del nodo 3
5. Escritura generada por el procesador del nodo 0

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

| **P1** | **P2** |
|---|---|
| `x=1;` | `y=1;` |
| `x=2;` | `y=2;` |
| `print y ;` | `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 resultados posibles.

**Ejercicio 5.** Supongamos que se va a ejecutar en paralelo el siguiente código (inicialmente *x* e *y* son 0):

| **P1** | **P2** |
|---|---|
| `x=30;`<br>`y=40;`<br>`flag=1;` | `while (flag==0) {};`<br>`r1=x;`<br>`r2=y;` |

Qué datos puede obtener P2 en r1 y r2 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 con un modelo de consistencia que relaja todos los órdenes en los accesos a memoria. Razone su respuesta.

**Ejercicio 6.** Se quiere implementar un cerrojo simple en un multiprocesador SMP basado en procesadores de la línea x86 de Intel, en particular, procesadores Intel Core. **(a)** Teniendo en cuenta el modelo de consistencia de memoria que ofrece el hardware de este multiprocesador ¿podríamos implementar la función de liberación del cerrojo simple usando "`mov k, 0`", siendo `k` la variable cerrojo? Razone su respuesta. **(b)** ¿Cómo se debería implementar la función de liberación de un cerrojo simple si se usan procesadores Itanium? Razone su respuesta.

**Ejercicio 7.** Se ha ejecutado el siguiente código en un multiprocesador con un modelo de consistencia que no garantiza ni W->R ni W->W (garantiza el resto de órdenes):

```
(1) sump = 0;

(2) for (i=ithread  ; i<8  ; i=i+nthread) {
(3)       sump = sump + a[i];
    }
(4) while (Fetch_&_Or(k,1)==1)  {};
(5) sum = sum + sump;
(6) k=0;
```

Conteste a las siguientes preguntas (considere que el compilador no altera el código):

**(a)** Indique qué se puede obtener en `sum` si se suma la lista `a`={1,2,3,4,5,6,7,8}. `k` y `sum` son variables compartidas que están inicialmente a 0 (el resto de variables son privadas), `nthread` = 3, `ithread` es el identificador del thread en el grupo (0,1,2). Si hay varios posibles resultados, se tienen que dar todos ellos. Justifique su respuesta.

**(b)** ¿Qué resultados se pueden obtener si lo único que no garantiza el modelo de consistencia es el orden W->R? Justifique su respuesta.

**Ejercicio 8.** ¿Qué ocurre si en el segundo código para implementar barreras visto en clase eliminamos la variable local, `cont_local`, sustituyéndola en los puntos del código donde aparece por el contador compartido asociado a la barrera `bar[id].cont`?

**Ejercicio 9.** Suponiendo que la arquitectura dispone de instrucciones `Fetch&Add`, simplifique el segundo código para barreras visto en clase.

**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`.

**Ejercicio 11.** Un programador está usando el siguiente código para barreras (`bar` es un vector compartido, `k` es una variable compartida, el resto son variables locales, `Fetch_&_Or(k,1)` realiza sus accesos a memoria antes de que puedan realizarse los accesos posteriores):

```
Barrera(id, num_procesos)
{
(1)    band_local= !(band_local)
(2)    while (Fetch_&_Or(k,1)==1) {};
(3)    cont_local = ++bar[id].cont;
(4)    k=0;
(5)    if (cont_local == num_procesos)  {
(6)       bar[id].cont=0;
(7)       bar[id].band=band_local;
       }
(8)    else while (bar[id].band != band_local) {};
}
```

Conteste a las siguientes cuestiones (considere que el compilador no altera el código):

**(a)** ¿Funciona bien este código como barrera en un multiprocesador en el que lo único que no garantiza su modelo de consistencia es el orden W->R? Razone por qué.

**(b)** Funciona bien este código como barrera en un multiprocesador con modelo de consistencia de memoria de ordenación débil? Razone por qué.

**Ejercicio 12.** Se quiere implementar un programa que calcule en paralelo la siguiente expresión en un multiprocesador en el que sólo se relaja el orden W->R y en el que sólo se dispone de primitiva de sincronización `test_&_set`:

$$d = \frac{1}{N}\sum_{i=1}^{N} x_i^2 - \bar{x}^2 \text{, donde } \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Un programador ha implementado el código de abajo. Tenga en cuenta lo siguiente: el código lo ejecutan `nthread` threads en paralelo; `ithread` es una variable local que nota el identificador del thread; `i`, `medl` y `varil` son variables locales; `i`, `med`, `vari`, el vector `x` y `N` son variables compartidas; inicialmente `med`, `vari`, `medl` y `varil` son 0.

```
(1)  for (i=ithread;i<N;i=i+nthread) {
(2)    medl=medl+x[i];
(3)    varil=varil+x[i]*x[i];
(4)  }
(5)  med = med + medl/N; vari = vari + varil/N;
(6)  vari= vari - med*med;
(7)  if (ithread==0) printf("varianza = %f", vari); //imprime en pantalla
```

Conteste a las siguientes cuestiones (considere que el compilador no altera el código):

**(a)** Se ha ejecutado este código usando varios threads y se ha visto que, aunque `N` y el vector `x` no varían, no siempre se imprime lo mismo. ¿Por qué ocurre esto?

**(b)** Añada lo mínimo necesario para solucionar el problema teniendo en cuenta que sólo se dispone para implementar sincronización de `test_&_set` (tampoco se dispone de primitivas software de sincronización). Indique qué variables son ahora compartidas y cuáles locales.

**(c)** Escriba el programa suponiendo que el multiprocesador además tiene primitivas de sincronización `fetch_&_add` (se tendrá en cuenta las prestaciones). Indique qué variables son compartidas y cuáles locales.

**(d)** Escriba el programa ahora suponiendo que el multiprocesador sólo tiene primitivas de sincronización `compare_&_swap` (se tendrá en cuenta las prestaciones). Indique qué variables son compartidas y cuáles locales.

NOTA: En todos los apartados puede añadir o quitar variables si lo estima conveniente.

**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: se queda con 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/Extended-Asm.html#Extended-Asm (http://gcc.gnu.org/onlinedocs/)

## 2 Cuestiones

**Cuestión 1.** Diferencias entre núcleos con multithread temporal y núcleos con multithread simultánea.

**Cuestión 2.** Diferencias entre núcleos con multithread temporal de grano fino y núcleos con multithread temporal de grano grueso.

**Cuestión 3.** Suponga un multiprocesador con protocolo MESI de espionaje. Si un controlador de cache observa en el bus un paquete de petición de lectura exclusiva de un bloque que tiene en estado C, debe (indique cuál sería la respuesta correcta y razone por qué es la respuesta correcta):

a) Generar un paquete de respuesta con el bloque y pasar el bloque a estado I.
b) Pasar el bloque a estado I.
c) Generar un paquete de respuesta con el bloque y pasar el bloque a estado E.
d) No tiene que hacer nada

**Cuestión 4.** Suponga un multiprocesador con protocolo MESI de espionaje. Si un nodo observa en el bus un paquete de petición de lectura exclusiva de un bloque que tiene en estado M, ¿qué debe hacer? Razone su respuesta.

**Cuestión 5.** Suponga un multiprocesador con el protocolo MESI de espionaje. Si el procesador de un nodo escribe en un bloque que tiene en su cache en estado I, debe (indique cuál sería la respuesta correcta y razone por qué es la respuesta correcta):

a) Generar paquete de petición de acceso E al bloque y pasar el bloque a M
b) Generar paquete de petición de acceso E y pasar el bloque a estado E
c) Generar paquete de petición de acceso E con lectura y pasar el bloque a estado E
d) Generar paquete de petición de acceso E al bloque con lectura y pasar el bloque a M

**Cuestión 6.** ¿Cuál de los siguientes modelos de consistencia permite mejores tiempos de ejecución? Justifique su respuesta.

a) modelo de consistencia que no garantiza los órdenes W->W y W->R
b) modelo implementado en los procesadores de la línea x86
c) modelo de consistencia secuencial
d) modelo de consistencia que no garantiza ningún orden.

**Cuestión 7.** Indique qué expresión no se corresponde con la serie (justifique su respuesta):

```
a) Lock
b) Fetch_and_Or
c) Compare_and_Swap
d) Test_and_Set
```