Examen

Nombre: Julián Bayardo Spadafora

Libreta Universitaria: 850/13

Ejercicio 1

Para todos los casos, puede simplemente correrse los ejemplos en herd7. La forma fácil de hacer esto es utilizando los archivos **regen.sh** en cada uno de los directorios de tso, pso, y ppc. El script se ocupa de correr los tests y generar grafos para los casos en los que la condición sucede; los archivos de extensión png que quedan en el directorio contienen las dependencias para que el caso suceda en el modelo dado; si el archivo no existe, entonces el ejemplo no sucede.

A continuación, una explicación más elaborada/intuitiva de por qué cada ejemplo sucede o no en los modelos pedidos.

SC

1. No sucede. Si se ejecuta cualquier escritura de la línea 1, la lectura en la siguiente línea será 1 para alguno de los dos threads.
2. Sucede. Load Buffering puede suceder en SC si cada uno de los threads corre la línea 1 antes de que alguno corra la línea 2.
3. No sucede. La escritura de los registros sucede como primera instrucción, y en ambos casos toma el valor en memoria, que es 0.
4. MFENCE no es una operación válida dentro de este modelo. Si borramos la instrucción, se vuelve exactamente el ejemplo 1, que no puede suceder.
5. Ídem 4.
6. No sucede. Observemos que:
    a. Si r3=0, la instrucción r3=x tiene que haber ejecutado antes que x=1
    b. Para que r2=1, la instrucción r2=y tiene que ejecutar antes que y=1
    c. Para que r1=1, la instrucción r1=x debe suceder después que x=1

    Es decir, hay un ciclo.

7. No sucede. El argumento es igual al caso anterior. Hacer el grafo con todos los órdenes sobre el programa no es acíclico.
8. MFENCE no es una operación válida dentro de este modelo. Si borramos la instrucción, se vuelve exactamente el ejemplo anterior.
9. Ídem 8.

TSO

TSO incluye las relajaciones en SC y más, por lo que todo lo que sucede en SC sucede en TSO, implicando que no hay por qué cubrir el punto 2.

1. Sucede. Dado que no hay un flush de los buffers entre las instrucciones, la lectura de **x** e **y** en ambos procesadores puede ser el valor en memoria, que es inicialmente 0.
2. Ídem SC.

3. No sucede. La escritura de los registros sucede como primera instrucción, y en ambos casos toma el valor en memoria, que es 0.
4. Sucede. Esto es porque si bien P0 tiene orden entre sus MOV, P1 no los tiene. El siguiente orden de ejecución generaría ese caso:
    a. P1 ejecuta entero, pero no realiza un fence en ningún momento. En cuyo caso, **EBX** tiene el valor 0, e **y** tiene el valor 1 pero está dentro del buffer, que aún no es llevado a memoria principal.
    b. Ejecuta todo P0, en cuyo caso **EAX** termina con el valor 0, ya que P1 todavía no hizo la escritura de 1 a **y** en la memoria.
    c. P1 tiene un flush del buffer (eventualmente).
5. No sucede. Los buffers son flusheados en ambos casos: no importa quién comience la ejecución, alguno de los dos va a ejecutar un fence luego de un MOV, lo que forzará al otro a tener el valor 1 en la escritura de registro.
6. No sucede. Observemos que la única forma de que esto suceda es que el P2 esté de alguna forma "fuera de sincronización" con la escritura de P0, pero no con la de P1. El problema es que para que la escritura de P1 suceda, antes debe haber sucedido su lectura, y si un procesador ve una escritura (P1 vería la escritura de **x** en el ejemplo), todos los otros también la ven.
7. No sucede, y el motivo es el mismo que en el caso anterior. Observemos que en este caso los procesadores "ven cruzado": P2 ve a P0 pero no a P1, y P3 ve a P1 pero no a P0. Para que la condición suceda, ambos procesadores deben ejecutar la primera instrucción después del write que están leyendo, pero antes del write que no. Esto genera un ciclo en el grafo.
8. No sucede. Las lecturas producidas por P2 y P3 están cruzadas y generan un ciclo.
9. Ídem 8.

PSO

Sabemos que PSO es TSO con una relajación adicional[1]: escrituras a distintos lugares de memoria pueden ser reordenados entre sí (es decir, el store buffer deja de ser FIFO, y pasa a ser otro mecanismo). Para poder correr los ejemplos en herd7, podemos modificar el modelo de TSO agregando la relajación.

Con la idea de verificar que el cambio está dentro de lo razonable, agregué el test PSO-TSO-REORD; el mismo verifica si hay reordenamiento de writes a distintos lugares de memoria. Además, agregué el test PSO-TSO-NOREORD para verificar que writes a los mismos lugares dentro de un mismo procesador no sean reordenados.

Si bien estos test no son indicadores de que el modelo escrito sea correcto con respecto a la especificación de PSO, sí muestran que sucede el reordenamiento de writes. Intuitivamente, es posible que haya problemas en el modelado con respecto a los writes iniciales (el conjunto IW en herd); pero no me queda claro cómo verificar esto.

Con respecto a los comportamientos, ninguno de los ejemplos tiene casos donde haya más de un write a una posición de memoria dentro de un mismo procesador, por lo que las respuestas son iguales a las de TSO (observemos que el comportamiento de TSO sucede cuando el store buffer de PSO se comporta

[1] http://15418.courses.cs.cmu.edu/spring2013/article/41, https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-823-computer-system-architecture-fall-2005/lecture-notes/l20_relaxedmm.pdf

como si fuera FIFO sólo por accidente; esta ejecución es una de las tantas permitidas). Utilizar herd muestra efectivamente lo mismo.

PowerPC

Todos estos casos fueron analizados utilizando los ejemplos del emulador online PCCMEM[2] referido en clase (estos archivos no están incluidos en la entrega, pero son accesibles online). El test respectivo en PCCMEM se encuentra entre paréntesis en cada ejemplo; además, los test se encuentran con el prefijo PPC- en el directorio HERD/tests.

1. Sucede (SB). PPC puede reordenar las lecturas de x e y locales en cada thread para que sucedan primero.
2. Sucede (LB). Mismo motivo que en SC.
3. No sucede. Esto está arquitecturalmente prohibido en PPC[3].
4. Sucede (SB+sync+po). PPC podría reordenar la ejecución de P1 para que el MOV a **EBX** suceda antes que el MOV a **y**, y además podría suceder que P0 ejecute entero entre medio del MOV a **EBX** y el MOV a **y** reordenado.
5. No sucede (SB+syncs, cuidado porque la condición en el test por defecto es ~exists). Tiene sentido porque si eso sucediese, las escrituras a registro podrían ser reordenadas antes del fence, y el fence se supone que sincroniza la memoria.
6. Sucede (WRC).
7. Sucede (IRIW). En clase vimos el ejemplo de cómo los early writes y early reads pueden resultar en dicha ejecución.
8. Sucede. El cambio es irrelevante con respecto al ejemplo anterior, ya que los SYNC podrían ser pausados hasta el final de la ejecución de P2 y P3 y sería exactamente lo mismo (y esto es una ejecución posible).
9. No sucede (IRIW+syncs, cuidado porque la condición del test por defecto es ~exists).

Ejercicio 2

Los subejercicios 1 y 2 se encuentran en la carpeta “java”, mientras que los 3 y 4 en la carpeta “cpp”. Para compilar los ejemplos de C++ es más fácil usar el script **compile.sh**, este requiere el paquete cmake3.

[2] http://www.cl.cam.ac.uk/~pes20/ppcmem/

[3] http://www.cl.cam.ac.uk/~pes20/MMmeet-2014-09-slides/slides-MMmeet-thin-air-2014-09-23.pdf página 6.