Arquitectura e Ingeniería de Computadores

Modulo III. Multiprocesadores

Tema 8. Coherencia. **Consistencia.** Sincronización.

Consistencia de memoria

- Esquema
  - Motivación.
  - Modelo de consistencia de memoria.
  - Consistencia Secuencial (CS).
  - Implementación CS de alto rendimiento.
  - Modelos relajados.
  - Resumen.

Motivación

- **Coherencia**
  - Garantiza que la escritura de una posición de memoria se hace visible a todos los potenciales lectores en el mismo orden
    - La escritura de A se hace visible a P_2, en el mismo orden que P_3…
- **¿Es suficiente?**
  - La coherencia No dice nada sobre el orden con el que se hacen visibles las escrituras con relación al acceso a otras posiciones de memoria
  - ¿Cuándo se debe realizar una escritura?
  - Necesitamos algo más que la coherencia para dar a un espacio de direcciones compartido una semántica clara

Modelo de Consistencia de Memoria (1)

Consistencia

Modelo de ordenación que los programadores puedan usar para razonar acerca de la corrección de los programas

Contrato entre el programador y el diseñador del sistema

- **Modelo de Consistencia de la Memoria**
  - **Especifica las restricciones en el orden en el que las operaciones de memoria deben hacerse visibles a los procesadores**
  - Los programadores se basan en el modelo de consistencia para razonar acerca de los posibles resultados (corrección de los programas)
  - Para el diseñador del compilador o del hardware el modelo impone limitaciones a posibles reordenaciones (optimizaciones)

Modelo de Consistencia de Memoria (2)

- **Sistemas Uniprocesador**
  - El orden en el cual deben parecer haberse ejecutado los accesos a memoria es el orden secuencial especificado por el programador

Orden de programa

  - Si el HW o el SW alteran el orden de las instrucciones, para mejorar las prestaciones, debe detectar y resolver las dependencias de datos y de control para que parezca que no se ha alterado:
    - Para las instrucciones que no presentan dependencia de datos no hay ninguna restricción respecto al orden de su ejecución.
    - Para las instrucciones que presentan dependencia de datos hay mecanismos HW y SW para garantizar que se ejecutan en el orden del programa.
      - Algoritmo tomasulo
      - Buffer de reordenamiento….

Modelo de Consistencia de Memoria (3)

- **Ejemplo 1**

Uniprocesador

P

…
(1) A = 1;
(2) Copia = A;
…

- Los mecanismos HW/SW aseguran que se ejecuta la instrucción (1) antes que (2)

Modelo de Consistencia de Memoria (4)

- **Ejemplo 1**

Multiprocesador

P_1	P_2
…	…
(1)A = 1;	(2)Copia = A;
…	…

- No se realiza control de dependencia de datos entre operaciones de acceso a memoria de distintos procesadores
  - No se asegura que se ejecute la instrucción (1) antes que (2)

El programador No puede esperar que el resultado del programa en paralelo coincida con el del programa secuencial

Modelo de Consistencia de Memoria (5)

- **Solución:**
  - El programador tiene que **usar algún código extra para sincronización**

Multiprocesador

P_1	P_2
…	…
A = 1;	**While K=0 {}**
K=1;	Copia = A;
…	…

Necesidad de definir de forma más precisa el modelo de consistencia

- **Tampoco hay seguridad de que el código se ejecute correctamente**
  - Sólo se garantiza evitar los problemas de dependencia de datos entre operaciones de acceso a memoria que se ejecutan en cada procesador
  - El SW/HW del procesador P1 puede alterar el orden de ejecución de sus instrucciones porque no son dependientes

Modelo de Consistencia de Memoria (6)

- **Modelo de consistencia en multiprocesadores**
  - Especifica restricciones en el orden en que las operaciones de memoria se hacen visibles a los procesadores, las unas respecto a las otras, incluyendo:
    - Operaciones a la misma posición o a distintas posiciones de memoria
    - Realizadas por un mismo procesador o por diferentes procesadores.
  - Consistencia incluye a la Coherencia

Modelo de Consistencia de Memoria (7)

- **Principales alternativas:**
  - **Consistencia secuencial (SC)**
    - Extensión natural del modelo de consistencia uniprocesador
  - **Modelos Relajados**

Consistencia Secuencial (1)

- El modelo es similar al entrelazado de threads dentro de un único procesador de tiempo compartido:
  - Dentro de un mismo proceso
    - El orden es el que impone el programa
      - Aunque exista ejecución fuera de orden, la finalización es en orden
  - Entre procesos
    - Cualquier intercalado de los procesos sería válido
      - Respetando que para cambiar de proceso, las instrucciones del primero finalicen

Consistencia Secuencial (2)

Consistencia Secuencial (Lamport 1979)

Un multiprocesador es secuencialmente consistente si el resultado de la ejecución de cualquier programa es la mismo que el que se obtendría si las operaciones de cada procesador individual aparecieran en el orden especificado por el programa y las operaciones de todos los procesadores se realizaran según algún orden secuencial

- En cada procesador se debe mantener el orden entre operaciones que indique el código que ejecuta → **orden de programa** **(orden parcial)**
- Globalmente se debe mantener un orden secuencial entre todas las operaciones → **atomicidad** **(orden total)**
  - **Ejecución atómica** → la operación parece globalmente (a todos los procesadores) que se ejecuta y se completa antes de que empiece la siguiente.
    - Todas las escrituras se ven por todos los procesadores en el mismo orden.

Consistencia Secuencial (3)

- No elimina la necesidad de **usar mecanismos de sincronización**
  - Dado que permite que se entrelacen las instrucciones de distintos procesadores, el orden no está determinado
  - Si el programador quiere acotar a un solo resultado, debe establecer restricciones en el orden de entrelazamiento
- **Mecanismos de Sincronización** → Necesarios para establecer un orden global determinado

Consistencia Secuencial (4)

- Ejemplo 1
  - **(a) SC sin sincronización**
    - No se realiza control de dependencia de datos entre operaciones de acceso a memoria de distintos procesadores
      - No se asegura que se ejecute la instrucción (1) antes que (2)

(a)

P_1	P_2
…	…
(1)A = 1;	(2)Copia = A;
…	…

(b)

P_1	P_2
…	…
A = 1;	While K=0 ¨{}
K=1;	Copia = A;

El programador puede esperar que el resultado del programa en paralelo coincida con el secuencial

  - **(b) SC + sincronización**
    - El SW/HW del P1 **no puede alterar el orden de ejecución de sus instrucciones** aunque no son dependientes
    - **Ejecución atómica**: globalmente no se ejecuta una instrucción hasta que no se ha completado la anterior

Consistencia Secuencial (5)

- Ejemplo 2

```
P1                    /*Valor inicial de A y B = 0*/        P2
(1a) A = 1;                                                (2a) print B;
(1b) B = 2;                                                (2b) print A;
```

- Resultados posibles bajo **CS**
  - **Orden parcial** del programa =>
    - En P1: 1a → 1b
    - En P2: 2a → 2b
  - **Orden global** del programa => varias posibilidades

orden	Resultado
1a → 1b → 2a → 2b	(A,B): (1,2)
1a → 2a → 1b → 2b	(1,0)
1a → 2a → 2b → 1b	(1,0)
2a → 1a → 2b → 1b	(1,0)
2a → 1a → 1b → 2b	(1,0)
2a → 2b→ 1a → 2b	(0,0)

Con los mecanismos de sincronización el programador se encarga de dar un orden global determinado para obtener el resultado deseado

Consistencia Secuencial (6)

- Ejemplo 2

```
P1                    /*Valor inicial de A y B = 0*/        P2
(1a) A = 1;                                                (2a) print B;
(1b) B = 2;                                                (2b) print A;
```

- El resultado (0,2) no es posible bajo consistencia secuencial
  - A = 0 implica 2b->1a, bajo CS implica que 2a->1b
  - B = 2 implica 1b->2a, **contradicción**

Consistencia Secuencial (7)

- Restricciones que debe cumplir el HW multiprocesador para implementar el modelo de SC:
  - **El orden parcial debe ser el orden de programa**
    - Ni el procesador ni el sistema de comunicaciones pueden desordenarlo
    - El procesador
      - La ejecución es en orden
      - NO permite muchas optimizaciones del compilador y del procesador
    - El sistema de comunicaciones
      - En bus compartido se implementa de manera sencilla
      - En redes punto a punto:
        - No se garantizan que los paquetes lleguen a sus destinos en el mismo orden que se han inyectado a la red
        - Para garantizar el orden del código un procesador **no podrá realizar un nuevo acceso a memoria hasta que reciba el reconocimiento de actualización o invalidación** de cada uno de los procesadores con copia en su cache

Consistencia Secuencial (8)

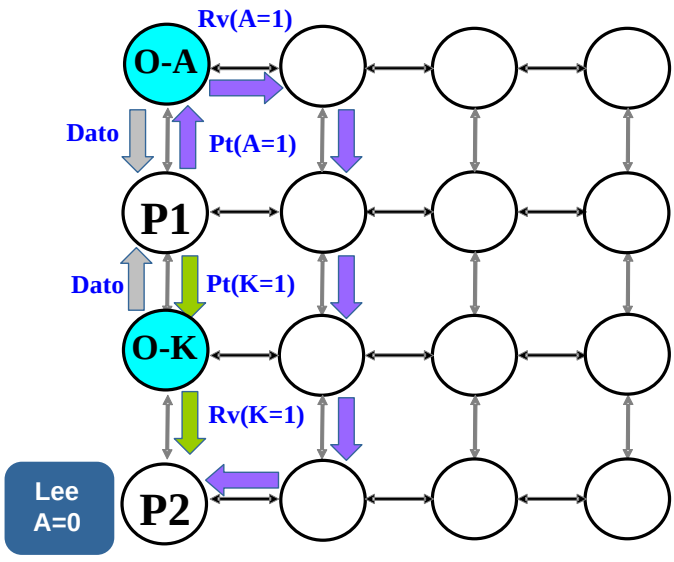
- Volviendo al ejemplo 1

P_1	P_2
A = 1;	while K=0 {};
K = 1;	Copia = A;

-Valor inicial de A y K = 0
-P2 y P1 tienen una copia de K y A

Para mantener orden de programa

- P1 no podrá realizar un nuevo acceso a memoria hasta que reciba el reconocimiento de actualización o invalidación de cada uno de los procesadores con copia



Consistencia Secuencial (9)

- Restricciones que debe cumplir el HW multiprocesador para implementar el modelo de SC:
  - **Orden total → atomicidad. Para ello hay que:**
    - **Serializar las escrituras de una misma dirección**
    - **Serializar las escrituras de distintas direcciones**

Consistencia Secuencial (10)

- **Serializar las escrituras de una misma dirección**
  - Lo proporciona el **modelo de coherencia**
  - En bus compartido el orden de las escrituras es el orden de acceso al bus
  - En redes punto a punto:
    - El nodo origen es el que envía las actualizaciones/invalidaciones a los procesadores que comparten dato
      - El orden de escrituras lo impone el orden de acceso al directorio
    - Fácil de implementar en invalidación porque la siguiente lectura a esa dirección produce fallo
    - Para implementar en validación el nodo origen **no debe enviar la actualización de la siguiente escritura hasta que no recibe el reconocimiento de actualización de cada uno de los procesadores con copia**, de la escritura actual

Consistencia Secuencial (11)

- Ejemplo 3

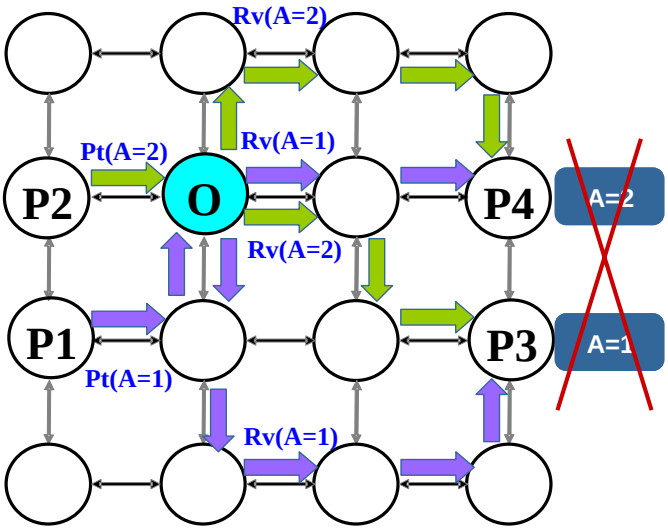
Valor inicial de A B y C = 0. P3 y P4 tienen copia de A

P_1	P_2	P_3	P_4
A = 1;	A = 2;	while B!=1 {};	while B!=1 {};
B = 2;	C = 1;	while C!=1 {};	while C!=1 {};
		Registro1=A;	Registro2=A;

Para serializar las escrituras de una misma dirección

-El Nodo origen manda la actualización

-El nodo origen no envía la actualización de la siguiente escritura hasta que no recibe el reconocimiento de actualización de P3 y P4



Consistencia Secuencial (12)

- **Serializar las escrituras cualquier dirección**
  - **Debe prohibirse la lectura de un nuevo valor escrito hasta que esta escritura sea visible a todos los procesadores**
  - En bus compartido el orden de las escrituras es el orden de acceso al bus
  - En redes punto a punto:
    - Puede haber accesos a distintos directorios por ser diferentes direcciones por lo que el orden NO lo impone el acceso a directorio
    - Fácil de implementar en invalidación porque la siguiente lectura a esa dirección produce fallo
    - Complejo en validación porque:
      - La confirmación de validación sólo la reciben el nodo que ha solicitado la escritura
      - Las caches con copia reciben la actualización pero no saben cuando se han actualizado todas

Consistencia Secuencial (13)

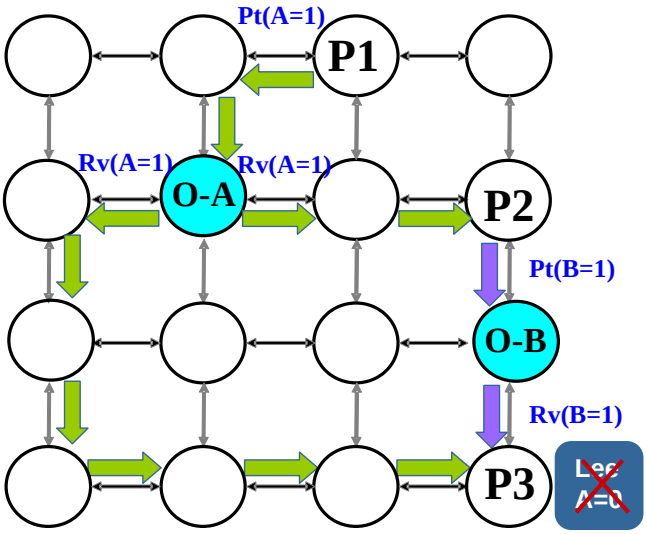
- Ejemplo 4

Valor inicial de A y B = 0. P2 y P3 tienen copia de A y B

P_1	P_2	P_3
A = 1;		
	If (A = 1);	
	B = 1;	
		If (B=1)
		Registro = A;

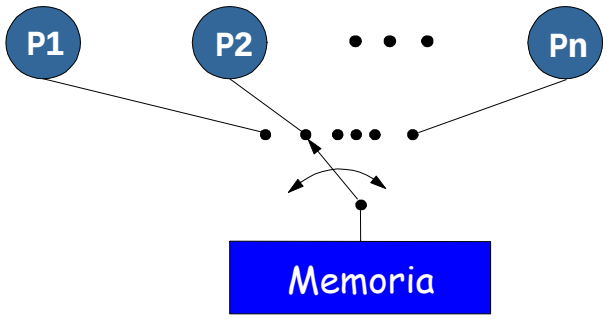
Para serializar las escrituras de distintas direcciones

- P2 no puede leer en A hasta recibir confirmación de que todas las caches con copia se han actualizado



Consistencia Secuencial (14)

- **Para el programador**
  - El sistema de memoria se comporta como una memoria central global conectada a los procesadores a través de un conmutador central
    - Cada procesador emite instrucciones en el **orden de su programa**.
    - El conmutador una vez atendido un acceso a memoria de un procesador conecta la memoria con otro procesador, **elegido al azar,** para atender la petición (**atomicidad)**
    - Del orden en que se eligen no se preocupa la consistencia, lo hace la sincronización.



  - El modelo de programación **es simple e intuitivo**
    - Valen los razonamientos de multiprogramación de tiempo-compartido
    - Pero no permite muchas optimizaciones del compilador y del procesador

Consistencia Secuencial (15)

- **Condiciones suficientes para CS**
  - **Condiciones locales:**
    - Cada procesador emite las operaciones siguiendo el orden del programa
    - En caso de escritura, el procesador debe esperar a que **se complete** antes de emitir la siguiente operación de memoria
  - **Condición global**
    - Antes de una lectura, el procesador que quiere emitirla debe esperar a que **se complete** la escritura que produce el valor

NOTA: Se considera que una operación de memoria se ha completado cuando ha finalizado para todos los procesadores.

¿Demasiado restrictivo?

Condiciones Suficientes ≠ Condiciones Necesarias

Consistencia Secuencial (16)

- Volviendo al ejemplo 2

```
P1                    /*Valor inicial de A y B = 0*/        P2
(1a) A = 1;                                          (2a) print B;
(1b) B = 2;                                          (2b) print A;
```

- Resultados posibles bajo **CS**
  - **Orden parcial** del programa =>
    - En P1: 1a → 1b
    - En P2: 2a → 2b
  - **Orden global** del programa => varias posibilidades

orden	Resultado
1a → 1b → 2a → 2b	(A,B): (1,2)
1a → 2a → 1b → 2b	(1,0)
1a → 2a → 2b → 1b	(1,0)
2a → 1a → 2b → 1b	(1,0)
2a → 1a → 1b → 2b	(1,0)
2a → 2b→ 1a → 2b	(0,0)

Con los mecanismos de sincronización el programador se encarga de dar un orden global determinado para obtener el resultado deseado

Consistencia Secuencial (17)

- Volviendo al ejemplo 2

```
      P1        /*Valor inicial de A y B = 0*/        P2
(1a) A = 1;                                   (2a) print B;
(1b) B = 2;                                   (2b) print A;
```

  - Si el programador elige como resultado del programa el (1,2), existen dos posibles órdenes para obtenerlo:
    - La ejecución **1a → 1b → 2a → 2b**
      - Si cumple el modelo de CS: P1 no ejecuta en el orden de programa
    - La ejecución **1b → 1a → 2a → 2b**
      - **No cumple el modelo de CS**: P1 no ejecuta en el orden de programa
  - No importa que las operaciones se hayan completado fuera de orden si el resultado es el mismo

Implementación CS de alto rendimiento (1)

- Objetivo
  - "Suavizar" el orden total manteniendo el resultado
- Idea
  - Orden total SÓLO en caso de dependencia de datos
    - **Posibilidad de fuera de orden**
- ¿Cómo llevarla a la práctica?
  - Es necesario detectar dependencias
    - Entre procesadores
    - Dentro de un mismo procesador
  - El mecanismo de coherencia permite "aislar" los procesadores
    - En ausencia de actividad de coherencia (fallos de cache y/o invalidaciones) se sabe que no hay dependencias entre procesadores.
    - **Basarse en la actividad de coherencia**

Implementación CS de alto rendimiento (2)

- Detectar las dependencias
  - Globalmente → Entre procesadores
    - **Ordenar los fallos de cache e invalidaciones**
      - Bus común: arbitraje.
      - Red punto-a-punto: mensajes de confirmación
  - Localmente → Dentro de un mismo procesador
    - **Ordenar los accesos locales** entre los limites impuestos por los fallos de cache e invalidaciones.
      - Se pueden reordenar los accesos usando una LSQ
- Problema
  - **Es preciso ordenar los accesos locales con respecto a la actividad de coherencia** (fallos/invalidaciones)
    - Usar mecanismos que permitan **especulación**

Modelos relajados de consistencia (1)

- Clave:
  - **Sólo las referencias que implican sincronización deben ser ordenadas**
  - **Es posible relajar las restricciones de orden para el resto**
    - Permite una implementación fuera-de-orden de alto rendimiento
  - **Se puede relajar también la atomicidad**

Modelos relajados de consistencia (2)

- Es necesario que el programador marque/etiquete las referencias de sincronización
- El programador debe evitar los problemas derivados de las relajaciones en la restricciones,
  - utilizando instrucciones que ofrezca el HW para ello.
- Esquemas:
  - Operaciones explícitas de sincronización
  - Barreras de memoria
    - Todas la operaciones de memoria anteriores deben finalizar antes de que comience la siguiente
    - Normalmente, en los computadores fuera-de-orden modernos, las barreras de memoria paran el pipeline hasta que se hayan completado las referencias anteriores

Resumen (1)

- La coherencia no es suficiente
- Modelo de consistencia de memoria
  - Modelo de ordenación que los programadores puedan usar para razonar acerca de la corrección de los programas
- **El modelo de consistencia influye en las prestaciones y en la facilidad de programar**
  - Equilibrio entre prestaciones y la complejidad de la programación

Resumen (2)

- Consistencia secuencial
  - Ventaja
    - Simple para el programador (similar multiprogramación)
  - Desventajas
    - Demasiado restrictivo, puede limitar el rendimiento
  - Implementación de alto rendimiento
    - Basada en la actividad de coherencia
    - Ordenación especulativa
- Modelos relajados
  - Solución más común: memory fence/barriers
  - Ventajas
    - Mayor rendimiento
    - HW más simple
  - Desventajas
    - Trasladan dificultad al programador
    - Si abundan las barreras, el rendimiento merma