Ingeniería de los Computadores

Sesión 13. Memoria (multiprocesadores)

Conceptos

- Clasificación de multiprocesadores atendiendo a la distribución de memoria
  - UMA (Uniform Memory Access)
  - NUMA (Non-Uniform Memory Access)
  - COMA (Cache-Only Memory Architecture)

Conceptos

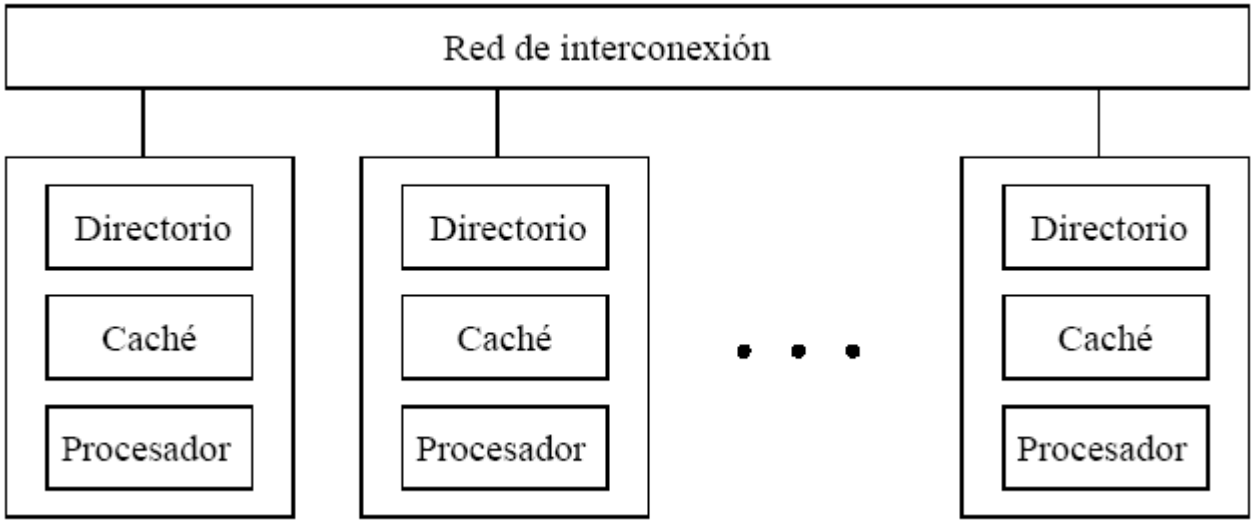
- Tipo UMA
  - Memoria centralizada, uniformemente compartida entre procesadores (todos los procesadores tienen el mismo tiempo de acceso a todas las palabras de la memoria)
  - Sistema fuertemente acoplado: alto grado de compartición de recursos (memoria) entre los procesadores dependencia funcional entre ellos
  - Cada procesador puede disponer de caché
  - Periféricos compartidos entre procesadores
  - Aplicaciones de propósito general y de tiempo compartido por múltiples usuarios
  - Sincronización y comunicación entre procesadores utilizando variables compartidas
  - Acceso a memoria, periféricos y distribución de procesos S.O.:
  - Equitativo →Symmetric (shared-memory) Multi Processors (SMPs)
  - No equitativo → Asymmetric (shared-memory) Multi Processors (AMPs)
  - CC-UMA (Caché-Coherent Uniform Memory Access)

- Tipo NUMA

  - Memoria compartida con tiempo de acceso dependiente de la ubicación de procesadores
  - Sistema débilmente acoplado: cada procesador dispone de una memoria local a la que puede acceder más rápidamente
  - Sistema de acceso global a memoria: local, global, local de otros módulos
  - CC-NUMA (Caché-Coherent Non-Uniform Memory Access) → Memoria compartida distribuida y directorios de caché

  - Ventajas
    - Escalado de memoria con + coste/rendimiento
    - Reducción de latencia de acceso a memorias locales

- Tipo COMA
  - Sólo se usa una caché como memoria
  - Caso particular de NUMA donde las memorias distribuidas se convierten en cachés
  - Las cachés forman un mismo espacio global de direcciones
  - Acceso a las cachés por directorio distribuido



- Consistencia de memoria

Problema: Transmisión de información entre procesadores en memoria compartida

Ejemplo:

/* El valor inicial de A y flag es 0 */

P1	**P2**
A=1	while (flag == 0); /*bucle vacío*/
flag=1	print A

P2 ha de realizar una espera activa hasta que *flag* cambie a '1'.

Después imprimirá A = '1' (suponiendo que en P1 *"A=1"* es una instrucción anterior a *"flag=1"*) → ¿COHERENCIA DEL SISTEMA DE MEMORIA? Escrituras en orden de programa.

Pero … y si "flag=1" se ejecuta antes de "A=1" ¿?

Es necesario un **modelo de consistencia de la memoria** en un espacio de direcciones compartido. Su objetivo es especificar restricciones en el orden de las operaciones en la memoria y proporcionar una visión uniforme de las mismas a los demás procesadores, logrando una abstracción de las operaciones de memoria independiente de la localización de las operaciones de memoria (módulos) y de los procesos involucrados.

Conceptos

- Consistencia de memoria
  - "Un modelo de consistencia de memoria especifica el orden en el cual las operaciones de acceso a memoria deben **parecer** haberse realizado (operaciones de lectura, escritura)"
  - En un procesador (o sistema uniprocesador), el orden en el que deben parecer haberse ejecutado los accesos a memoria es el orden secuencial especificado por el programador (o la herramienta de programación en el código que añade), denominado orden de programa.
  - Tanto el hardware como la herramienta de programación si que pueden alterar dicho orden, para mejorar las prestaciones, pero debe parecer en la ejecución del programa que no se ha alterado.

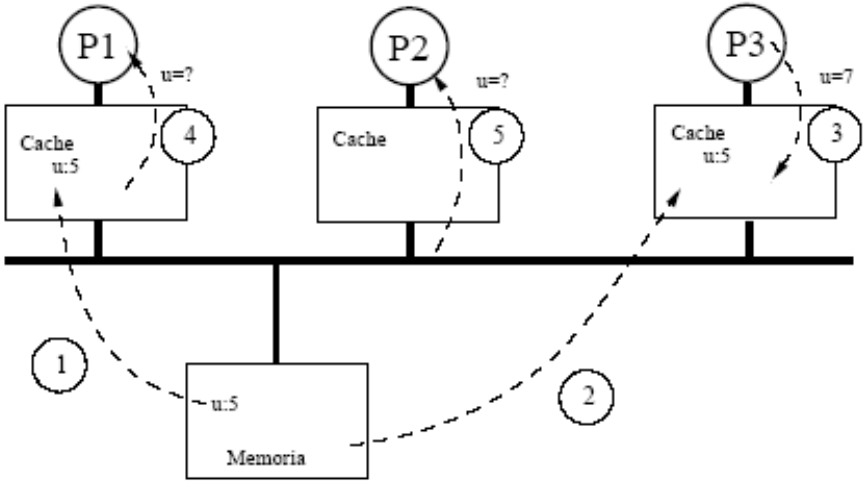
Conceptos

- Consistencia de memoria. Se debe garantizar que:
  - Cada lectura de una dirección, proporcione el último valor escrito en dicha dirección **(dependencia de datos lectura después de escritura)**
  - Si se escribe varias veces en una dirección se debe retornar el último valor escrito **(dependencia escritura después de escritura)**
  - Si se escribe en una dirección a la que previamente se ha accedido para leer, no se debe obtener en la lectura previa el valor escrito posteriormente en la secuencia del programa especificada por el programador **(dependencia escritura después de lectura)**
  - No se puede escribir en una dirección si la escritura depende de una condición que no se cumple **(dependencias de control)**

- Ejemplo 1: 2P, caché write-through (cada vez que se escribe una línea de caché, se actualiza la memoria principal)

Secuencia	Acción	Caché A	Caché B	X en MP
1	CPU A lee X	1	-	1
2	CPU B lee X	1	1	1
3	CPU A escribe 0 en X	0	1	0

- Ejemplo 2: 3P, caché write-through / write-back (se actualiza la memoria principal escribiendo todo el bloque cuando se desaloja de la caché == aún más problemática)



- Un sistema de memoria es coherente si cualquier lectura de un dato devuelve el valor más reciente escrito de ese dato
- Aspectos críticos del sistema de memoria compartida: los datos devueltos por una lectura (coherencia) y cuándo un valor escrito será devuelto por una lectura (consistencia)
- Un sistema de memoria es coherente si cumple:
  - Preservación del orden del programa: una lectura por un procesador P de una posición X, que sigue a una escritura de P a X, sin que ningún otro procesador haya escrito nada en X entre la escritura y la lectura de P, siempre devuelve el valor escrito por P.
  - Visión coherente de la memoria: una lectura por un procesador de la posición X, que sigue a una escritura por otro procesador a X, devuelve el valor escrito si la lectura y escritura están suficientemente separados y no hay otras escrituras sobre X entre los dos accesos.
  - Serialización de operaciones concurrentes de escritura: Las escrituras a la misma posición por cualquiera dos procesadores se ven en el mismo orden por todos los procesadores.

- Un sistema de memoria multiprocesador es coherente si el resultado de cualquier ejecución de un programa es tal que, para cada localización es posible construir una hipotética ordenación secuencial de todas las operaciones realizadas sobre dicha localización que sea consistente con los resultados de la ejecución y en el cuál:

  1. Las operaciones emitidas por un procesador particular ocurren en la secuencia indicada y en el orden en el que dicho procesador las emite al sistema de memoria

  2. El valor devuelto por cada operación de lectura es el valor escrito por la última escritura en esa localización en la secuencia indicada

- Dos estrategias para abordar la coherencia de las cachés:
  - Resolución software: el compilador y el programador evitan la incoherencia entre cachés de datos compartidos.
  - Hardware: transparente al programador mediante la provisión de mecanismos hardware, esta es la solución más utilizada para mantener la coherencia.
- Políticas para mantener la coherencia:
  - Invalidación de escritura o coherencia dinámica (write invalidate): siempre que un procesador modifica un dato de un bloque en la caché, invalida todas las demás copias de ese bloque guardadas en las otras cachés de los procesadores.
  - Actualización en escritura (write-update ó write-broadcast): esta política lo que hace es actualizar la copia en las demás cachés en vez de invalidarla.

- Dos estrategias para abordar la coherencia de las cachés:
  - Resolución software: el compilador y el programador evitan la incoherencia entre cachés de datos compartidos.
  - Hardware: transparente al programador mediante la provisión de mecanismos hardware, esta es la solución más utilizada para mantener la coherencia.
- Políticas para mantener la coherencia:
  - Invalidación de escritura o coherencia dinámica (write invalidate): siempre que un procesador modifica un dato de un bloque en la caché, invalida todas las demás copias de ese bloque guardadas en las otras cachés de los procesadores.
  - Actualización en escritura (write-update ó write-broadcast): esta política lo que hace es actualizar la copia en las demás cachés en vez de invalidarla.