AC 21/22 - Tema 3: Arquitecturas con paralelismo a nivel de thread TLP

1. Introducción

Este capítulo se centra en arquitecturas que permiten ejecutar en paralelo múltiples flujos de instrucciones (threads/hebras) que comparten memoria, o sea, arquitecturas con paralelismo a nivel de hebra con una única instancia del SO. El SO controla los flujos de instr.

Los modelos de programación paralela que siguen el paradigma de **variables compartidas** se implementan más fácilmente en arquitecturas TLP. Sin embargo, los que siguen el paradigma de **paso de mensajes** se acercan más a las arquitecturas TLP con múltiples instancias del SO o multicomputadores.

Hay partes del tema que NO aplican a los multicomputadores, como la coherencia del sist de memoria, consistencia del sist de memoria y sincronización de instr (apartados 3, 4, 5). En este tema **nos vamos a dedicar exclusivamente a las arquitecturas TLP con una única instancia del SO**.

2. Arquitecturas TLP

Las arquitecturas TLP con una sola instancia en el SO se pueden clasificar en...

- **Multiprocesadores:** ejecutan varios flujos de instr en paralelo con un computador con varios procesadores, cada flujo en un procesador o núcleo distinto. (Es decir, un multiprocesador puede encompasar un multinúcleo/multicore)
- **Multicore:** ejecutan varios flujos de instr en paralelo en un chip de procesamiento con múltiples núcleos, cada flujo en un núcleo distinto.

En los primeros multiprocesadores (años 60), el acceso a la memoria era uniforme (UMA, *uniform memory access*) - es decir, todos los procesadores tardaban lo mismo en acceder a cada posición de memoria. Para incrementar la escalabilidad, en los 90 se distribuyeron físicamente los módulos de memoria compartida entre los procesadores (NUMA, *Non uniform memory access*). Así se pudieron desarrollar configuraciones de cientos de procesadores manteniendo buenas prestaciones: aunque ya no tardan todos los núcleos el mismo tiempo en acceder a todas las posiciones de memoria, acceden más rápido a los módulos cercanos

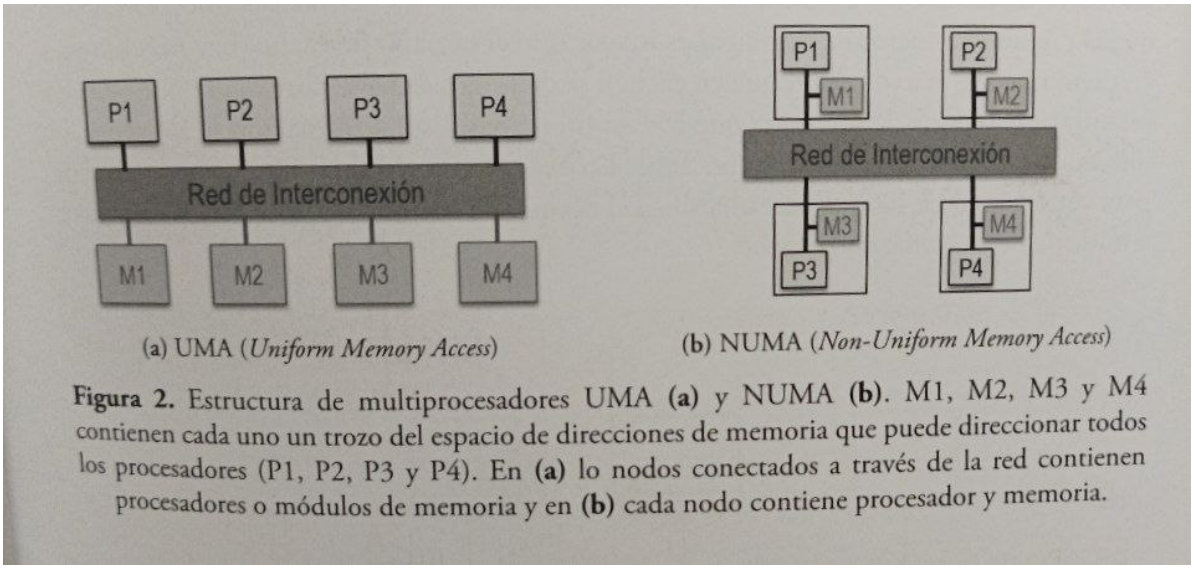


Figura 2. Estructura de multiprocesadores UMA **(a)** y NUMA **(b)**. M1, M2, M3 y M4 contienen cada uno un trozo del espacio de direcciones de memoria que puede direccionar todos los procesadores (P1, P2, P3 y P4). En **(a)** lo nodos conectados a través de la red contienen procesadores o módulos de memoria y en **(b)** cada nodo contiene procesador y memoria.

3. Coherencia del sistema de memoria

La presencia de cachés en un computador conlleva a que haya varias copias de una misma dirección de memoria: un dato puede estar en memoria principal, y tener una copia en la caché de un procesador o en varios. Cuando uno de los procesador escribe sobre su copia de caché, **se produce una incoherencia**, porque la copia no coincide con el contenido en MP, o el contenido de las cachés de otros núcleos.

Estas incoherencias son inadmitibles porque permiten el acceso a valores no actualizados de una dirección. Las situaciones de incoherencia se deben abordar, bien no permitiendo que sucedan, o evitando que causen problemas en caso de permitirse. Actualmente, las cachés implentan dos **mecanismos de actualización** de MP:

- **Escritura directa/*Write-through*:** no se permiten incoherencias porque cada vez que se escribe en la copia de un bloque de memoria, se escribe también en MP.

  En las aplicaciones de un ordenador es común que cuando escribamos en MP, pronto escribamos en las direcciones cercanas o de nuevo en la misma (principios de localidad espacial y temporal). Si es así, sería más rentable escribir el bloque completo en memoria tras múltiples escrituras en la caché.
- **Posescritura/*Write-back*:** se escribe en MP un bloque modificado en la caché cuando éste se desaloja para hacer sitio a otro bloque al que se quiere acceder. De esta forma se pueden escribir a la vez en memoria múltiples modificaciones de un bloque. Esto también nos ahorra actualizaciones en MP innecesarias (ej. no se usan inmediatamente por otros dispositivos o bloques). Para saber si un bloque en la caché ha sido actualizado, hay un bit de estado a la entrada del directorio de caché, que se pondrá a 1 cuando sea modificado.

Las cachés suelen usar posescritura, es decir, las incoherencias se permiten. Por esto, cuando se intente acceder a una [posición de memoria modificada en la caché de otro dispositivo], se debe suministrar la última actualización del bloque. Todo esto forma parte del **protocolo de mantenimiento de coherencia del sistema de memoria**.

Aunque, como vemos, se permiten las incoherencias caché-memoria, no se permiten las de caché-caché. Para garantizarlo, se deben cumplir dos condiciones:

- **Propagación de escrituras:** todo lo que se escribe en la copia de un bloque de una caché se propaga a las copias del bloque del resto de cachés.

    - **Escritura con actualización (*write-update*):** cada vez que un procesador escribe en una dirección de su caché, se escribe también en esa dirección en las demás copias. Si se cumplen los principios de localidad espacial y temporal, esto puede generar un tráfico innecesario.
    - **Escritura con invalidación (*write-invalidate*):** antes de que un procesador modifique una de sus direcciones de caché, se invalidan las copias del bloque que contienen la dirección en el resto de cachés. Esto permite que se pueda seguir escribiendo en el mismo bloque desde el primer procesador sin generar tráfico. Cuando otro procesador vaya a modificar la dirección, recibirá el bloque actualizado (desde la caché del procesador primero, o desde MP si está actualizada), propagando la escritura.
    - La propagación de escrituras puede realizarse:
        - **Difusión:** difusión de los paquetes de actualización o invalidación a todas las cachés
        - **Envío selectivo:** envío de los pauqetes de actualización o invalidación a las cachés con copia del bloque afectado.
            - Esto requiere llevar un **directorio de memoria** que informe de las cachés que contienen una copia del bloque. Este directorio tendría una entrada para cada bloque con info de su estado en el sistema de memoria e info de las cachés con copia del bloque.
- **Serialización de escrituras:** las escrituras en una dirección de memoria se ven en el mismo orden por todos los procesadores, deben parecer que realiza las modificaciones en serie, secuencialmente. Debe dar la impresión de que son atómicas.

En una estructura UMA se puede utilizar una red bus para las transferencias entre las cachés de los procesadores y la MP. La red bus implementa de forma natural una difusión, y todos los paquetes se ven por todos los elementos conectados al bus. Además, el orden en el que llegan al bus serializa los accesos (las escrituras). Todo esto hace que, en el caso de los UMA con red bus no sea necesario mantener información del resto de cachés y se pueda prescindir del directorio de memoria.

3.1. Protocolos de mantenimiento de coherencia

Para diseñar o describir un protocolo de mantenimiento de coherencia se debe especificar:

- **Política de actualización** de MP (posescritura o escritura directa)
- **Política de propagación** de escritura de una caché a otras (inmediata o invalidación)
- Comportamiento:
    - Posibles estados de un bloque en caché
    - Posibles estados de un bloque en MP
    - Paquetes que genera el controlador de caché ante eventos (lectura, escritura...)
    - Paquetes que genera el controlador de MP ante eventos (lectura, escritura...)
    - Relación de acciones con eventos y estados

Vamos a describir 4 protocolos de mantenimiento de coherencia, dos para UMA con bus y dos para NUMA en una placa.

Todos ellos usan posescritura como política de actualización de memoria, e invalidación como política de propagación de escritura a cachés.

3.1.1. Protocolo MSI (Modified-Shared-Invalid) de espionaje [UMA]

Los protocolos para buses se denominan de espionaje porque, al estar conectados todos los controladores al bus, pueden espiar los accesos del resto sondeando el bus. MSI es el protocolo con menos estados que utiliza posescritura e invalidación.

- **Estados de un bloque en caché:**
    - Modificado M - un bloque en este estado es la única copia válida del bloque en todo el sistema de memoria. Si el controlador ve en el bus una solicitud de lectura del bloque, mandará este; y si se reemplaza el bloque en la caché por otro, primero hay que mandar el bloque modificado a MP.
    - Compartido S - es válido, puede ser válido en MP y existir otras copias válidas en cachés.
    - Inválido I - o está en caché en estado inválido, o directamente no está en ninguna caché. Si se quiere acceder a esta dirección, se genera un fallo de caché y se hace una solicitud del bloque por el bus.
- **Estados de un bloque en memoria:**
    - Válido - está actualizado en MP, puede haber una o varias copias válidas en caché. Si el controlador de memoria ve una petición de lectura en el bus, envía un bloque válido.
    - Inválido - no está actualizado en MP, hay al menos una copia válida en caché.
- **Paquetes que genera el controlador de caché:** son los paquetes que se envían por el bus ante eventos en el procesador del nodo (lectura/escritura), el controlador de caché (reemplazo de bloque) o que llegan desde otros nodos. Son peticiones PT o respuestas RP. B = bloque.
    - PTlect(B) - Generado tras una lectura con fallo de caché en el procesador del nodo (el bloque que queremos leer no está en la caché).
        - RPbloque(B) - Lo recibe como respuesta el anterior, desde MP o desde la caché que tenga el bloque en estado Modificado
    - PTlectEX(B) - Lectura exclusiva, se genera como consecuencia ante una escritura por parte del procesador (PRescr) en un bloque con estado INVÁLIDO.
        - Recibe RPbloque(B) de igual forma que en anterior caso y además...
        - Se invalidarán las copias existentes del bloque en el resto de cachés
    - PTEX(B) - Acceso exclusivo al bloque. Se genera por PRescr en un bloque con estado COMPARTIDO. Si el bloque fuera válido, pero en estado MOD, no lo necesitaría porque ya tiene acceso exclusivo.
        - Como respuesta se invalidan todas las copias del bloque menos la que vamos a tocar.
        - Existen implementaciones que no cuentan con esta instrucción - en tal caso, simplemente se hace una PTlecEX(B), y simplemente se desecha el paquete enviado como respuesta.
    - PTposescr(B) - Por el reemplazo de un bloque en estado MOD
    - RPbloque(B) - Como hemos visto, lo envía MP o, si existe, lo envía el procesador que contenga la copia en estado MOD.
- **Paquetes que genera el controlador de memoria:** genera paquetes de respuesta RP:
    - RPbloque(B) - Si se almacena el estado en memoria, el controlador genera este paquete cuando en el bus hay una petición de lectura (PTlect(B) o PTlectEX(B)) de un bloque B que está en estado VAL. Si no hay bit de estado en MP, siempre genera una respuesta

que, si es necesario, intercepta el controlador de caché con el bloque en estado MOD o SHR.

- **Relación de acciones con eventos recibidos y estados:**

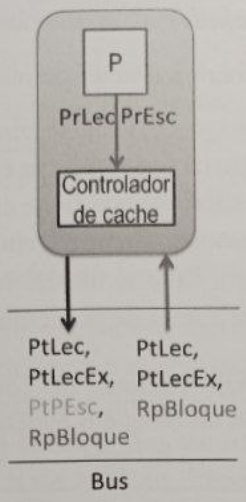


Figura 4. Paquetes generados por el controlador de caché de un nodo y eventos (paquetes) procedentes del exterior

Tabla 1. Protocolo MSI de espionaje. Acciones del controlador de caché de un nodo que provocan los eventos relacionados con un bloque B teniendo en cuenta el estado del bloque en la caché.

ESTADO ACTUAL de B	EVENTO	ACCIONES (del controlador de caché)	ESTADO SIGUIENTE de B
Modificado (M)	PrLec/ PrEsc(B)		Modificado
	PtLec(B)	- Genera paquete respuesta con B (RpBloque(B))	Compartido
	PtLecEx(B)	- Genera paquete respuesta con B (RpBloque(B)) - Invalida copia local	Inválido
	Reemplazo(B)	Genera paquete posescritura (PtPEsc(B)), que escribe el bloque B en memoria	Inválido
Compartido (S)	PrLec(B)		Compartido
	PrEsc(B)	- Genera paquete PtLecEx(B) (ó PtEx(B), si lo hubiera), para invalidar las copias de B en otras cachés	Modificado
	PtLec(B)		Compartido
	PtLecEx(B)	Invalida copia local del bloque B	Inválido
Inválido (I)	PrLec(B)	- Genera paquete PtLec(B), que provoca la respuesta con el bloque B (RpBloque(B)) de memoria o de la caché con el bloque en estado Modificado, si la hubiera. - Recoge RpBloque(B).	Compartido
	PrEsc(B)	- Genera paquete PtLecEx(B), que provoca (1) la invalidación de las copias del bloque B en otras cachés y (2) la respuesta con el bloque (RpBloque(B)) de memoria o de la caché con el bloque en estado Modificado, si la hubiera. - Recoge RpBloque(B).	Modificado
	PtLec/ PtLecEx(B)		Inválido

3.1.2. Protocolo MESI (Modified-Exclusive-Shared-Invalid) de espionaje [UMA]

Con el protocolo MSI siempre que vayamos a modificar un bloque, mandamos una petición de lectura exclusiva incluso si no hay en otras cachés copias del bloque que tengamos que invalidar. Cuando se ejecutan instrucciones secuenciales esto puede aumentar el tráfico considerablemente.

El protocolo MESI aborda este problema dividiendo el estado S (compartido) en dos: E (exclusivo, es la única copia) y S (compartido, hay al menos otra copia en otra caché).

Las diferencias con el MSI se marcan en negrita en la siguiente tabla:

Tabla 2. Protocolo MESI de espionaje. Acciones del controlador de caché de un nodo que provocan los eventos relacionados con un bloque B teniendo en cuenta el estado del bloque en la caché.

ESTADO ACTUAL de B	EVENTO	ACCIONES (del controlador de caché)	ESTADO SIGUIENTE de B
Modificado (M)	PrLec/ PrEsc(B)		Modificado
	PtLec(B)	- Genera paquete respuesta con B (RpBloque(B))	Compartido
	PtLecEx(B)	- Genera paquete respuesta con B (RpBloque(B)) - Invalida copia local	Inválido
	Reemplazo(B)	- Genera paquete posescritura (PtPEsc(B)), que escribe el bloque B en memoria	Inválido
Exclusivo (E)	PrLec		Exclusivo
	PrEsc		Modificado
	PtLec		Compartido
	PtLecEx	- Invalida copia local	Inválido
Compartido (S)	PrLec(B)		Compartido
	PrEsc(B)	- Genera paquete PtLecEx(B) (ó PtEx(B), si lo hubiera), para invalidar las copias de B en otras cachés	Modificado
	PtLec(B)		Compartido
	PtLecEx(B)	- Invalida copia local del bloque B	Inválido
Inválido (I)	PrLec(B) (hay cachés con copia de B)	- Genera paquete PtLec(B), que provoca la respuesta con el bloque B (RpBloque(B)) de memoria o de la caché con el bloque en estado Modificado, si la hubiera. - Recoge RpBloque(B).	Compartido
	PrLec(B) (no hay cachés con copia de B)	- Genera paquete PtLec(B), que provoca la respuesta con el bloque B (RpBloque(B)) de memoria. - Recoge RpBloque(B).	Exclusivo
	PrEsc(B)	- Genera paquete PtLecEx(B), que provoca (1) la invalidación de las copias del bloque B en otras cachés y (2) la respuesta con el bloque (RpBloque(B)) de memoria o de la caché con el bloque en estado Modificado, si la hubiera. - Recoge RpBloque(B).	Modificado
	PtLec/ PtLecEx(B)		Inválido

3.1.3. Protocolos MSI para multiprocesadores NUMA

Ahora vamos a describir dos protocolos MSI para multiprocesadores NUMA: uno difunde las peticiones a todos los nodos, y el otro sólo envía las peticiones al nodo que tiene en el bloque en el trozo de MP más cercano al nodo. Para describir estos protocolos vamos a distinguir tres tipos de nodos:

- Nodo solicitante de un bloque S - el que genera una petición del bloque (PTlect, PTEX, PTlectEX o PTescr).
- Nodo origen del bloque O - en los multiprocesadores NUMA la memoria está repartida físicamente en nodos, de forma que cada uno tiene en módulos de memoria más cercanos un trozo del espacio de direcciones total del multiprocesador. Podemos decir que estos módulos de memoria cercanos se hospedan en el nodo. El nodo O es aquel que tiene el bloque en el trozo de memoria que hospeda.
- Nodo propietario de un bloque P - es un nodo que tiene copia del bloque en su caché.

3.1.3.1. Protocolo MSI para multiprocesadores NUMA sin difusión

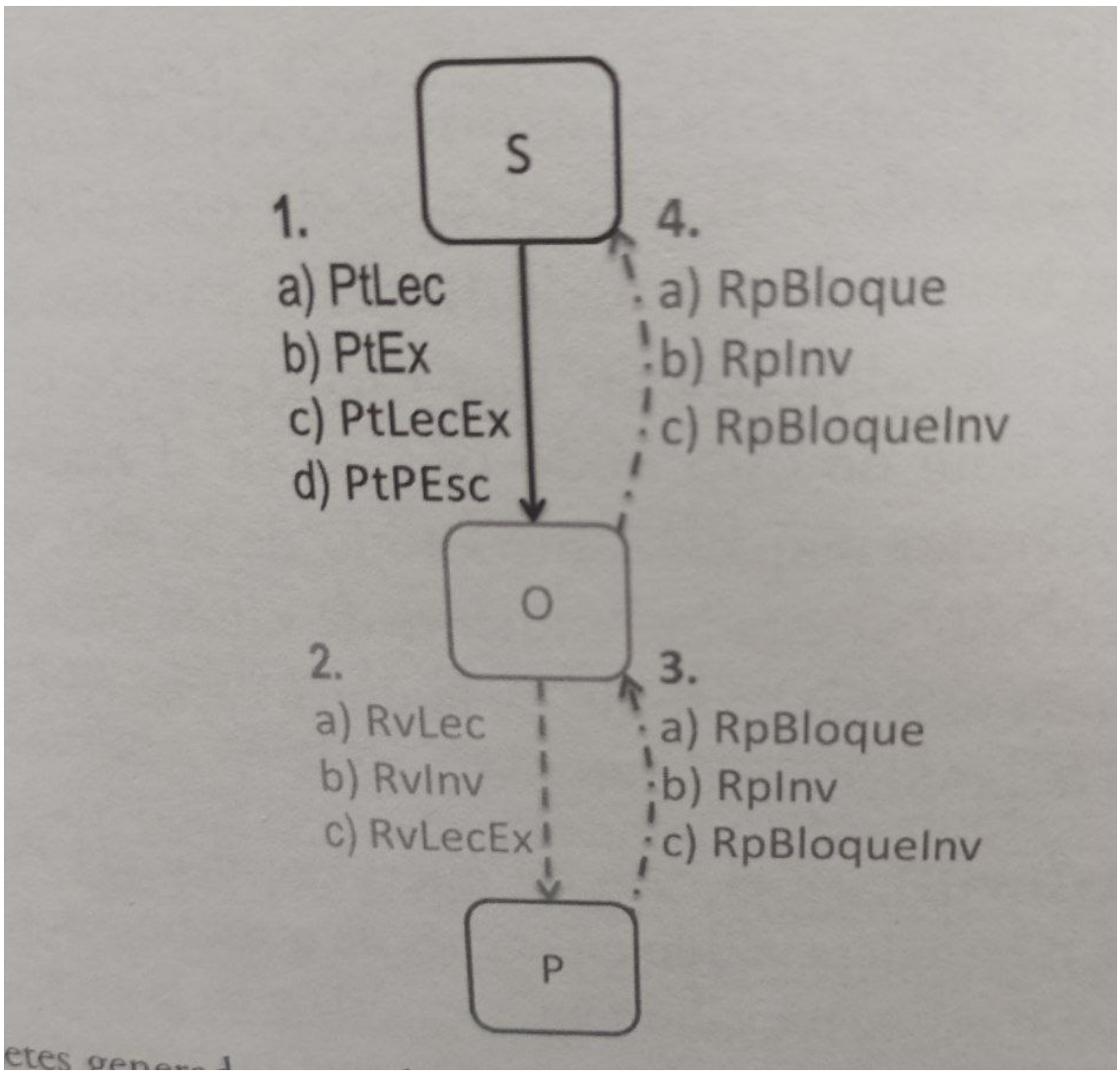
Como no se usa difusión, hay que [mantener en el directorio de MP] tanto el [estado del bloque de memoria] como [información sobre las cachés con copia del bloque], para que las invalidaciones solo vayan a los nodos con copia del bloque.

Bloque	**Estado memoria**	**Bits presencia para C0--C1--C2--C3**
B-1	V	0--0--0--0
B	V	1--1--0--1
B+1	I	0--0--0--1
...		

Se puede usar un vector de bits como el de arriba, en el que cada bit activo significa una caché en la que el bloque es válido. Todas las peticiones de un bloque se envían al nodo origen O del bloque, que es quien tiene el subdirectorio con información sobre el bloque. El nodo O propagará las invalidaciones a los nodos con copia válida, cumpliendo la condición de coherencia. También se cumple la condición de serialización.

- **Estados de un bloque en caché:** Modificado M, Compartido S, Inválido I.
- **Estados de un bloque en memoria:** Válido o Inválido. En algunos sistemas existen estados pendientes (Pendiente de válido, Pendiente de inválido), que se usan cuando está procesando el cambio. Si la memoria recibe una solicitud de un bloque en estado pendiente, responde con un reconocimiento negativo, para que el solicitante vuelva a intentarlo más tarde.
- **Paquetes generados por controladores de caché o de memoria y otras acciones:**
    - Paquetes de petición PT de...
        - Nodo solicitante S a nodo origen O:
            - PTLect(B), cuando se produce una lectura del procesador del nodo S con fallo de caché (no hay copia válida del bloque B en el nodo S). Cuando reciba la respuesta RPBloque(B) de O, la introducirá con estado SHR (compartido, no quiero repetir la S de solicitante y la de shared).

- PTEX(B), acceso exclusivo sin lectura en caso de producirse una escritura del procesador del nodo S en un bloque B que tiene en su caché en estado SHR. En tal caso no necesita pedir la lectura, porque tiene el bloque válido. Cuando reciba la respuesta confirmando la invalidación de la MP y las otras cachés RPinv(B), modificará el bloque y cambiará el estado a SHR.



3.1.3.2. Protocolo MSI para multiprocesadores NUMA con difusión

4. Consistencia del sistema de memoria

5. Sincronización

5.1. Cerrojos

5.2. Barreras

5.3. Instrucciones para implementar la sincronización