ARQUITECTURA E INGENIERÍA DE COMPUTADORES

Tema 2.6

Miquel Gómez Corral

Ing. Informática ETSINF, UPV

Tema 2.6

Procesadores Multinucleo

MULTIPROCESADOR

Son procesadores que tienen más de una CPU. ***CPU = Núcleo***. Cooperan entre ellos para ejecutar una o más aplicaciones y están **bajo un único sistema operativo** (O sea misma máquina, nada de un prosador aquí y otro allí).

Se **ejecuta en paralelo** cuando un mismo proceso tiene varios hilos, cuando hay varias aplicaciones secuenciales independientes o un mix con *hilos de un proceso y varios procesos indepes*.

Objetivo: Reducir el $T_{ejecución}$ de las aplicaciones y aumentar la productividad de instrucciones y programas.

Clasificación de los multiprocesadores

Las tareas tienen que sincronizarse y trabajar, pero también se han de poder pasar datos. Teniendo como criterio la comunicación/sincronización y La arquitectura de la memoria (todos a una o cada uno con la suya) tenemos:

Comunicación: **Variables compartidas** o **paso de mensajes**.

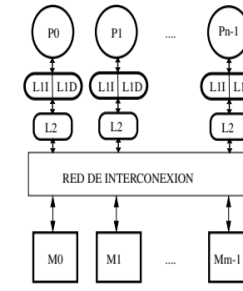
Arquitectura de la memoria: Los procesadores que tenga está **todos conectados a una misma memora** (***memoria compartida***) o **cada uno tiene una memoria** pa él (***memoria distribuida***).

Tipos: Multiprocesadores de **memoria compartida**, Multiprocesadores de **memoria compartida distribuida**, Multiprocesadores de memoria distribuida (clústers…)

Desde el punto de vista de la arquitectura es **mejor tener memoria compartida distribuida**. Si los tienes compartida pueden tener conflictos, y si es distribuida solo tarda mucho la información de uno a otro. Pero de esta manera el paso de información es muy rápido y además tienes más memoria.

MULTIPROCESADORES DE MEMORIA COMPARTIDA

Memoria compartida por todos los procesadores y **accesible desde cualquiera**. Cada procesador puede acceder a todo el espacio de direccionamiento. Mientras uno accede a una modulo concreto otro no puede, pero si quiere acceder a otro módulo de memoria SÍ pot.



Uniform Memory Access (*UMA*): Tiempo de acceso a un módulo es independiente del procesador, todos tardan lo mismo en acceder a cada modulo (siempre que no esté en uso).

Se usan muchas cachés para cada Procesador para que no pierda tiempo accediendo a las memorias y evitar que haya más interferencias entre ellos.

Comunicación y sincronización: La **comunicación es mediante variables compartidas** y la **sincronización mediante exclusión mutua** soportada por **instrucciones atómicas**. *El test and set… Tienes que hacer que la instrucción de comprobación se haga atómica, que en cada cosa que estás haciendo nadie se te meta por el medio.*

COHERENCIA ENTRE LAS CACHES

Si la cache de un procesador tiene un cache con un x, y otro procesador cambia esa variable. Pasan dos cosas, en **memoria principal no está cambiado** (*hay que cambiarlo para que los que vengan a leerlo después ya lo tengan write through, pero no es suficiente…*), y que al primero no se le actualiza el dato por arte de magia. Haría falta que se **actualizarán las variables**.

Soluciones dinámicas (*hardware*): **Detectan y resuelven el problema** en tiempo de ejecución. Se usan los Protocolos de coherencia.

Soluciones estáticas (*software*): **Evitan la aparición del problema**, impidiendo que algunas variables se puedan ubicar en cache, por lo que directamente no surgen conflictos, todo lo relacionado con esa variable tiene que pasar por la memoria principal. *Esto hace que sea más lento y en algunos casos no rente.* (*non-cacheables variable*).

PROTOCOLOS DE COHERENCIA

Protocolos de invalidación: Cuando **un procesador escribe sobre un bloque**, se les manda una "notificación" a todas la cache para que invaliden todos los bloques de memoria para que cuando alguien vuelva a ir a su caché a por el dato YA NO ESTARÁ y se irá a la memoria, consiguiendo así el valor actualizado.

Protocolos de actualización: **Igual que antes**, pero en vez de invalidar el valor aprovechan y van a memoria a leer el dato que se ha cambiado. *Se usa menos porque supone mucho uso de la memoria y la red de comunicación.*

Orden de coherencia: Los **"mensajes" que pasas para notificar** se llaman orden de coherencia. Cuando se actualiza un bloque de memoria, como todas las cachés están ahí escuchando a ver que hacen, les llega una difusión y **todos lo reciben**: **Protocolo Snoopy**. Si quieres que **SOLO lo reciban los que tienen en dato** te hacen falta **directorios**.

- **Hay 2 formas de hacer**: Si tienes pocos procesadores renta usar un bus. Si tienes bastantes procesadores, la otra forma es ver quién tiene una copia del bloque en su caché, y solo a ese le mandas el mensaje (xq a los demás no les sirve). ***Protocolo basado en directorios**, sitio donde está la lista de quién tiene qué bloques.*

Protocolo de coherencia (*MESI*) *Ole la ahí messiiii*

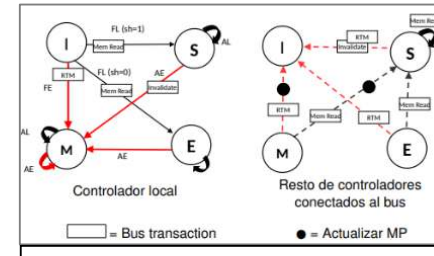
Pretende **minimizar la cantidad de mensajes** de invalidación/actualización. *Por ejemplo, si nadie más tiene el bloque.*

Autómata: Se le da a cada Bloque de **TU CACHÉ** un estado:

	Coherente con MP	Copia única
M (Modified)	No	Sí
E (Exclusive)	Sí	Sí
S (Shared)	Sí	Puede ser

M
E
S
I

- **E**: **Nadie lo ha tocado** y **Solo lo tienes tu**: *Léelo sin problema*.
- **M**: **No es coherente con MP** (*modificado*) Pero eres el **único que lo tiene**: *Modifícalo sin problemas*
- **S**: **No lo ha tocado nadie**, pero **puede ser que alguien más lo tengo**.
- **I**: **Sería inválido**, *que no lo tienes en TU cache*.



Al = acierto en lectura, AE = acierto escritura…
Los **puntos negros** son ESCRITURA EN MEMORIA PRINCIPAL

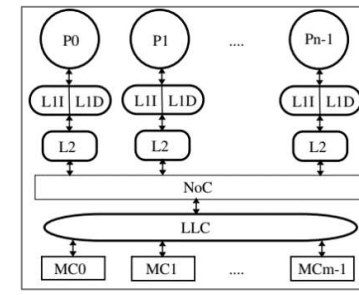
El estado de cada bloque en cada cache va pasando de un estado a otro según tu lees/modificas u otro lee o modifica.

Tuyas: fallo de lectura (FL), fallo de escritura (FE) y acierto de escritura (AE).

Otras caches (*mensajes del bus*): ***MemRead***, solicita el bloque a otra cache (*o a memoria*) ante un FL → Si alguna cache lo tiene pone en "shared", sino lo pongo en "E". ***Invalidat***, solicita invalidar un bloque para escribir ante un AE y ***ReadToModify***, requiere copia del bloque para modificarlo ante un FE

- Si no lo tenía y lo leo y soy el único: I -> E. Si vuelvo a leer E -> E.
- Si llega otro y lo lee: E -> S. Y él pasa de I -> S
- Si lo tenía y ahora lo modifico: E -> M o S -> M. Los demás que lo tengan pasarán del estado S -> I
- SI ahora otro de estos lo modifica sin tenerlo: I -> M, pero al primero pasara de M -> I
- Si estoy en modificado y lo vuelvo a modificas: M -> M

PROCESADORES MULTINÚCLEO

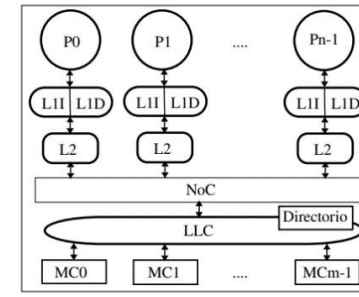


A **cada procesador** se le meten varios núcleos. En vez de poner un procesador con un núcleo más y más rápido (*que no es posible hacerlo más rápido a partir de cierto punto, xq no se hace mucho mejor*), pues ponemos varios núcleos.

Tonces ahora los núcleos de dentro de un procesador tienen que **organizar como se comunican**, igual que antes **Sincronizar accesos a memoria**…

Se les pone **más controladores de memoria** y **más cachés** (LLC las acces level caches). Eso y el NoC dejan que todos los núcleos puedan acceder al LLC.

DIRECTORIOS



Es donde se guarda que bloques tiene cada cache, por lo que cuando cambias un bloque, sabes a quién notificar. Reduces la cantidad de mensajes entre procesadores (caches) y el trafico en memoria.

Es un “Vector de presencia”, un array con información de quien tiene cada cosa.

Útil cuando hay muchos núcleos. No usan redes con buses o en anillo xq escalan mal.