DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343

Arquitectura de Computadores

Clase 14 - Paralelismo Avanzado y Coherencia de Caché

Profesor: Germán Leandro Contreras Sagredo

Objetivos de la clase

- Conocer la taxonomía de Flynn para entender distintos tipos de paralelismo.
- Conocer implementaciones que permitan un paralelismo real en el *pipeline*.
- Conocer los mecanismos de coherencia de caché en arquitecturas de memoria compartida.

Hasta ahora...

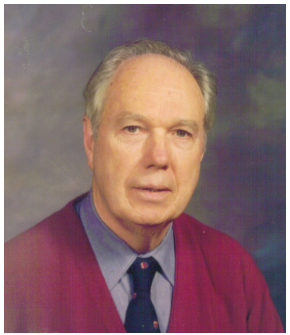
- Ya construimos un computador que tiene todo lo necesario para que pueda ser operado y programado, además de implementar lo necesario para que se comunique con otros dispositivos.

- Además, vimos cómo permitir la ejecución de múltiples procesos, cómo optimizar los accesos a memoria mediante cachés y cómo permitir la ejecución de múltiples instrucciones en un *pipeline*.

Ahora, veremos técnicas de paralelismo más avanzadas que permitan optimizar aún más los tiempos de ejecución de los programas.

Tipos de arquitectura paralela

Para poder categorizar y entender mejor cómo funcionan los distintos tipos de arquitecturas paralelas, hacemos uso de la **Taxonomía de Flynn**. Esta nos entrega cuatro tipos de paralelismo, según si este se aplica sobre los datos, sobre las instrucciones o ambos.



Michael J. Flynn (1934).
Propuso esta taxonomía en 1966.

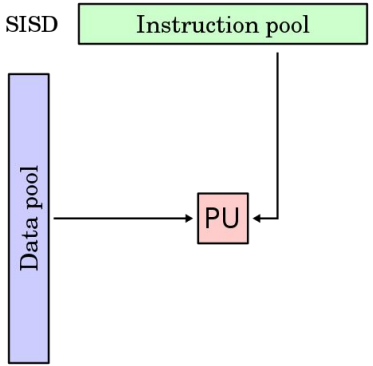
Paralelismo de instrucciones → Paralelismo de datos ↓	*Single Instruction*	*Multiple Instructions*
Single Data	**SISD**	**MISD**
Multiple Data	**SIMD**	**MIMD**

Tipos de arquitectura paralela - SISD

Single Instruction stream, Single Data stream

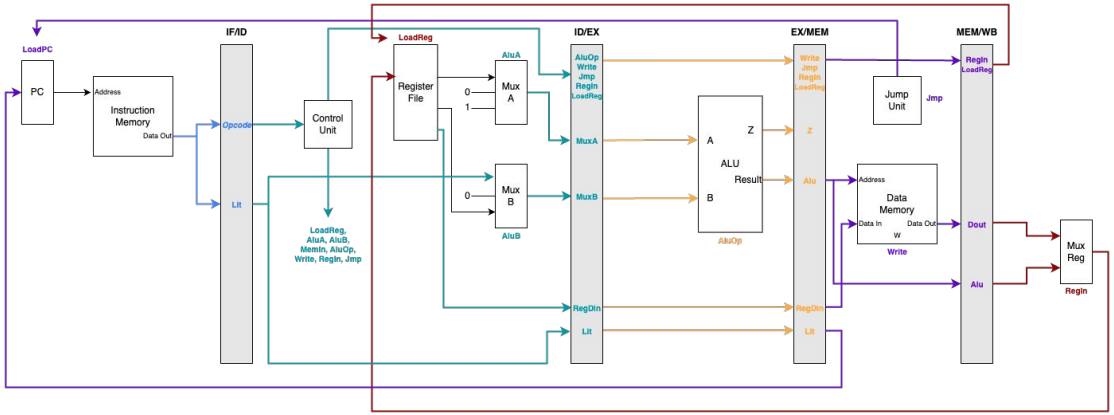
Corresponde a la generalización del paralelismo a nivel de instrucción (ILP). Un flujo de instrucciones opera sobre un flujo de datos en una única unidad de programación (PU).

¿Cómo podríamos extender el *pipeline* del computador básico para poder tener un paralelismo 100% real (*i.e.* la ejecución simultánea de dos instrucciones de inicio a fin)?



Tipos de arquitectura paralela - SISD

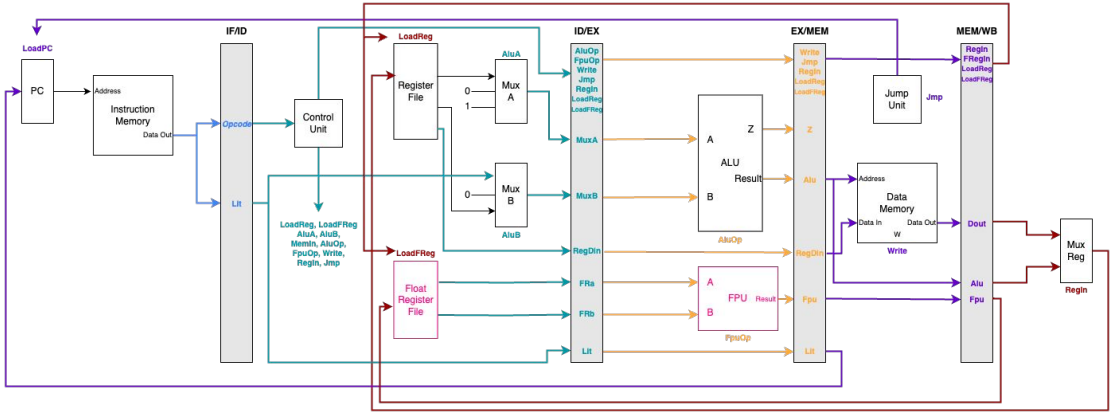
Primero, tomemos una versión reducida y generalizada de nuestro *pipeline* con los siguientes cambios:



- Usamos un *Register File* en vez de dos registros y de estos se seleccionan los que se operarán (como en RISC-V y como funciona en implementaciones reales).
- Nos olvidamos de los *hazards* y de cómo se resuelven para enfocarnos en cómo extender la paralelización.

Tipos de arquitectura paralela - SISD

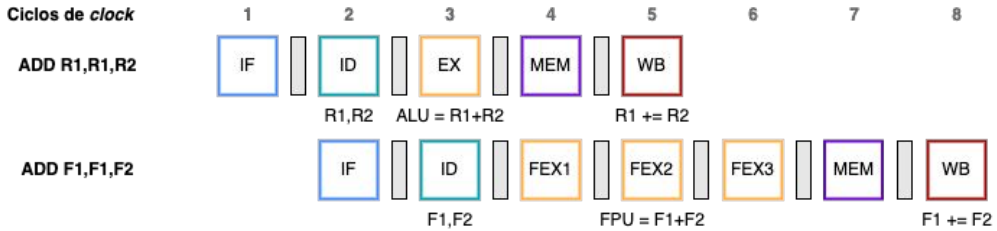
Ahora, extendemos la funcionalidad de nuestro *pipeline* agregando un segundo conjunto de registros y unidad de ejecución:



- Usamos un *Float Register File* y una FPU para operar con números de tipo *float* (similar a la extensión F de RISC-V). El resultado de la FPU se guarda en un registro *float* y no se toca la memoria de datos.
- De esta forma, podemos tener instrucciones que siguen un flujo **independiente** al que ya teníamos.

Tipos de arquitectura paralela - SISD

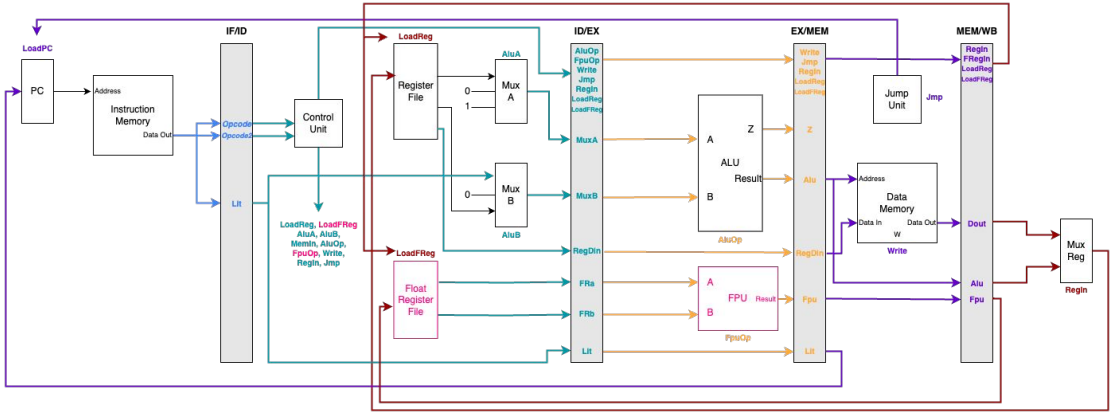
Supongamos que la ejecución de la FPU toma tres ciclos, por lo que el flujo de trabajo para dos instrucciones independientes se ve así:



En este caso, las etapas EX y WB son **independientes** ya que utilizan una unidad de ejecución y *Register File* distintos (MEM también, pero es un caso particular por ahora). ¿Cómo podemos extender las etapas IF e ID para poder ejecutar ambas instrucciones simultáneamente?

Tipos de arquitectura paralela - SISD

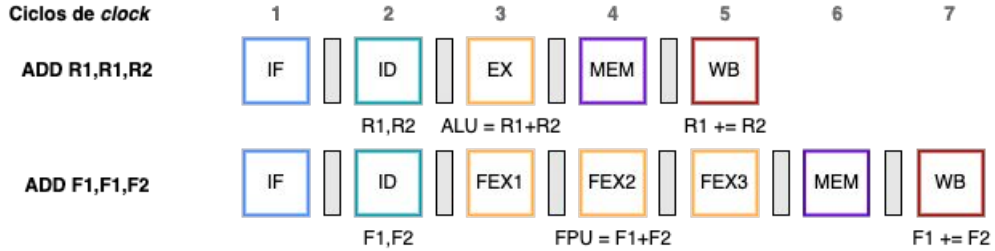
Podemos dar la capacidad de leer dos instrucciones contiguas y que la unidad de control sea capaz de decodificar ambas en un solo ciclo.



- El compilador tendría que asegurarse de que estas instrucciones queden intercaladas para evitar errores de ejecución.
- Esto trae consigo más *hazards* que resolver, pero de momento podemos asumir que se pueden resolver de forma similar a lo que ya hemos estudiado.

Tipos de arquitectura paralela - SISD

Con esta implementación, tenemos una CPU con paralelismo más cercano al 100%. Sigue sin serlo totalmente ya que solo se paralelizan instrucciones que son independientes entre ellas.



Lo que acabamos de hacer es transformar el procesador en uno ***multiple-issue***.

Tipos de arquitectura paralela - SISD

CPU *multiple-issue*: Procesador que permite obtener, decodificar y ejecutar múltiples instrucciones simultáneamente. Se califican de forma *N-issue*: si puede procesar 3 instrucciones simultáneamente, se considera 3-*issue*.

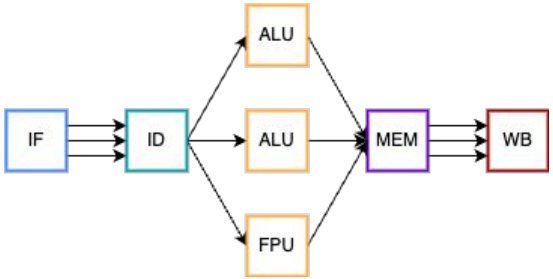


Diagrama de flujo de las etapas de un procesador 3-*issue*. En este caso, se puede asumir que se poseen 2 bancos de registros, 2 unidades de ejecución ALU y una FPU para llevar a cabo la implementación.

Extender la cantidad de accesos a memoria es complejo para direcciones no contiguas y no va a ser un problema a estudiar en este curso.

Tipos de arquitectura paralela - SISD

En nuestra implementación asumimos que el compilador se hace cargo del orden de las instrucciones para poder llevar a cabo el paralelismo. En la práctica, existen dos tipos de técnica para implementar un procesador *multiple-issue*.

- **Técnicas estáticas:** El compilador se encarga de agrupar las instrucciones para asegurar su ejecución paralela.

- **Técnicas dinámicas:** La CPU determina en tiempo de ejecución las instrucciones a paralelizar, despachándolas a unidades de ejecución distintas.

Tipos de arquitectura paralela - SISD

Técnica estática más utilizada: *Very Large Instruction Word* (VLIW)

En esta, el compilador genera un ***bundle*** o paquete de instrucciones que pueden ejecutarse en paralelo. Este paquete se envía como **una única instrucción muy larga**. Luego, la CPU se encarga de reordenarlas para ser despachadas de forma paralela.

Dirección	Instrucción
0x00	Instrucción 1
0x01	Instrucción 2
0x02	Instrucción 3
0x03	Instrucción 4
0x04	Instrucción 5
0x05	Instrucción 6
0x06	Instrucción 7
0x07	Instrucción 8
0x08	Instrucción 9

Compilador →

Dirección	Bundle			
0x00	Instrucción 1	Instrucción 6	Instrucción 7	NOP
0x01	NOP	NOP	Instrucción 3	Instrucción 4
0x02	NOP	Instrucción 2	NOP	NOP
0x03	NOP	Instrucción 5	Instrucción 9	NOP
0x04	NOP	NOP	NOP	Instrucción 8

Empaquetamiento de instrucciones realizado por el compilador. En vez de realizarse 9 *fetchs* de la memoria de instrucciones, se realizan solo 5 de forma que se puedan ejecutar paralelamente. En caso de no encontrar instrucciones válidas o de haber *hazards*, se insertan instrucciones NOP.

Tipos de arquitectura paralela - SISD

Técnica dinámica más utilizada: Arquitectura Superescalar

Esta se basa en dos componentes esenciales:

- ***Reservation Station***: Se encuentran antes de cada unidad de ejecución. Almacenan temporalmente las instrucciones **hasta que sus operandos estén disponibles** (por ejemplo, después de la etapa *Writeback*). En dicho instante despachan la instrucción para que realice su ejecución.
- ***Commit Unit* o *Reorder Buffer***: Almacena temporalmente todos los resultados de las unidades de ejecución y **los reordena** para evitar errores de dependencia de datos antes de despacharlos para su escritura en el banco de registros o memoria de datos.

Tipos de arquitectura paralela - SISD

Técnica dinámica más utilizada: Arquitectura Superescalar

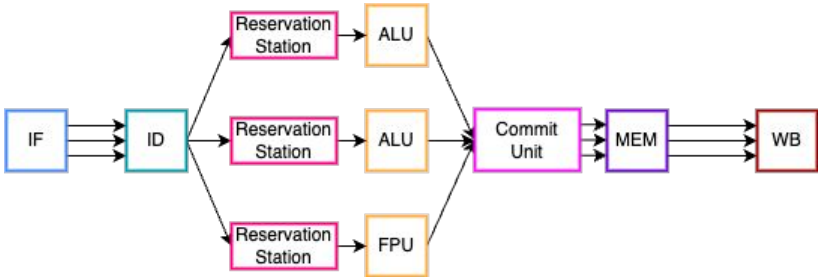


Diagrama de flujo de las etapas de un procesador 3-*issue* con arquitectura superescalar. Las implementaciones lógicas de las *Reservation Station* y *Commit Unit* son complejas y no nos centraremos en ellas en este curso.

Tipos de arquitectura paralela - SISD

Una de las grandes desventajas de las arquitecturas de tipo SISD es la complejidad que poseen para tener un buen rendimiento sin problemas de consistencia de datos. Esto, a su vez, implica un mayor costo monetario y energético para procesadores de estas características.

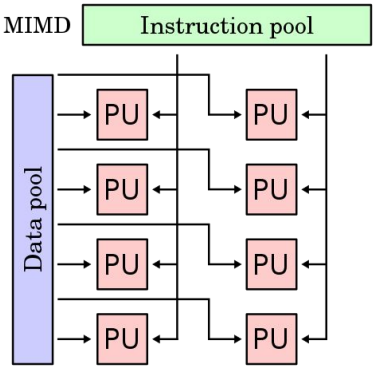
Otra alternativa, en base a estas complicaciones, es el **uso de múltiples procesadores más simples**.

Tipos de arquitectura paralela - MIMD

Multiple Instruction streams, Multiple Data streams

Múltiples flujos de instrucciones operan sobre múltiples flujos de datos a través de múltiples unidades de programación (PU).

Este tipo de arquitectura es ideal para la ejecución de **tareas independientes** (programas distintos, por ejemplo). Los sistemas que lo implementan se conocen como **sistemas multiprocesador**. Veremos dos categorías para estos sistemas.



Tipos de arquitectura paralela - MIMD

Sistema multiprocesador por paso de mensajes: En este tipo de arquitectura, cada procesador posee su propia memoria y se conecta con otros procesadores mediante una red. Si los procesadores están conectados en un mismo lugar físico con una red local, hablamos de un ***cluster***. Si están en lugares físicos distintos y conectados a través de Internet, entonces hablamos de un **sistema distribuido**.

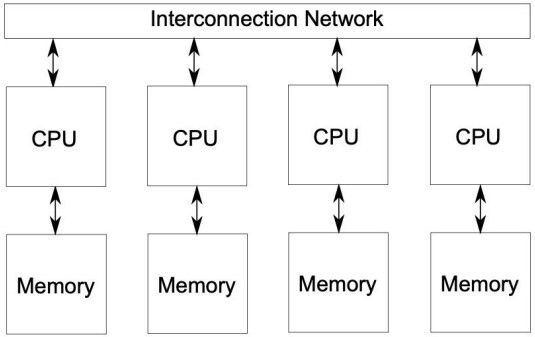


Diagrama general de un sistema multiprocesador por paso de mensajes.

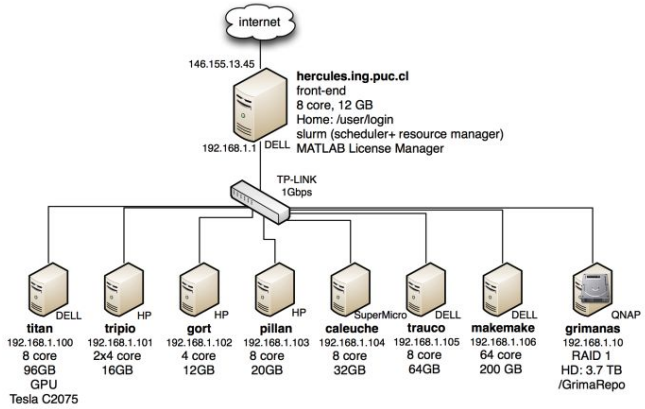


Diagrama del antiguo *cluster* del ex Grupo de Inteligencia de Máquina (GRIMA) del Departamento de Ciencia de la Computación UC. (Fuente: Cristian Ruz - IIC2523)

Tipos de arquitectura paralela - MIMD

Sistema multiprocesador con memoria compartida: En estos sistemas, los procesadores comparten en *algún nivel* su memoria, lo que facilita la transferencia de datos entre ellos. Cada uno podría tener su propia jerarquía de memoria (*i.e.* su propia caché) y compartirla en niveles más altos. Si todos los procesadores se incorporan en un único chip, hablamos de un procesador ***multicore***.

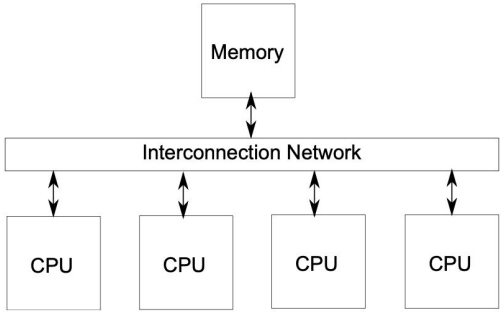
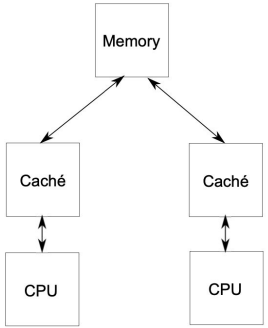
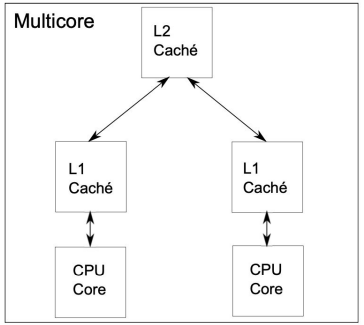


Diagrama general de un sistema multiprocesador con memoria compartida.



Memoria compartida en el segundo nivel, independiente en el primer nivel.

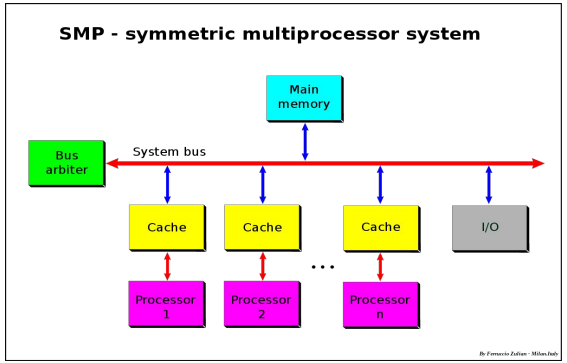


Procesador *multicore* con memoria compartida.

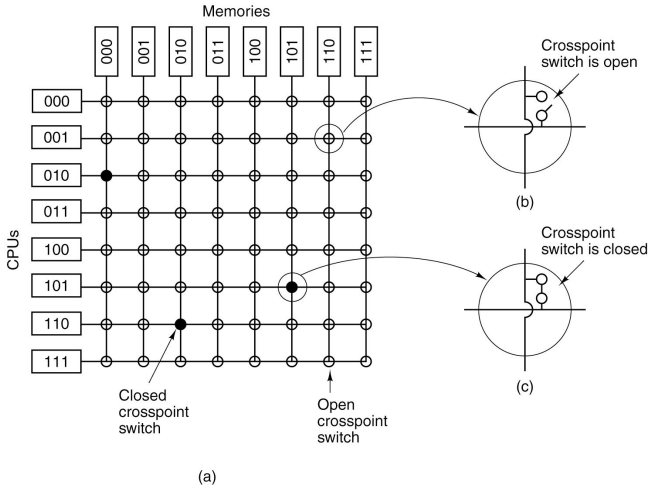
Tipos de arquitectura paralela - Memoria compartida

Esquemas de memoria compartida según tipo de acceso

Uniform Memory Access* (UMA):** Todos los procesadores acceden **de la misma forma** a la memoria. Puede ser a través de un único bus o un ***crossbar switch.



Ejemplo de arquitectura UMA:
Symmetric Multiprocessor System (SMP)



Con múltiples procesadores, el problema de múltiples accesos a memoria se vuelve relevante. En este caso, la memoria se puede dividir en distintos módulos y los *crossbar switches* habilitan el acceso de memoria al módulo si ningún procesador está accediendo a él.

Tipos de arquitectura paralela - Memoria compartida

Esquemas de memoria compartida según tipo de acceso

***Non-Uniform Memory Access* (NUMA):** Los procesadores se reparten en nodos donde cada uno puede acceder a un segmento específico de la memoria (memoria local). Pueden acceder a espacios de memoria asignados a otros nodos pero con una latencia de acceso mayor (memoria remota).

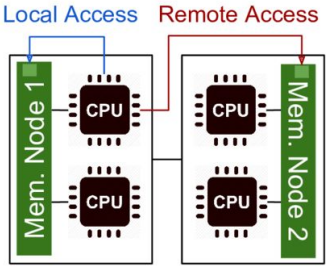


Diagrama general de un esquema NUMA.

Tipos de arquitectura paralela - Memoria compartida

En ambos esquemas de memoria compartida, se pueden agilizar los accesos a través de memorias cachés (con uno o más niveles).

Si las cachés siguen una política de escritura *write-through*, en general no habrá problemas ya que la memoria será consistente para todos los procesadores, pero cada escritura implica un *overhead* significativo que aumenta el tiempo de ejecución de los programas.

Por lo tanto, se opta por seguir el protocolo de escritura *write-back*, pero de este nace un nuevo problema: la **coherencia de caché**.

Coherencia de caché

Entendemos como **coherencia de caché** a la consistencia en la memoria caché de dos procesadores distintos que leen o escriben el dato de una misma dirección de memoria. Si uno de ellos escribe en esta dirección de memoria bajo el protocolo *write-back*, debemos asegurar de alguna manera que el otro procesador que acceda a este dato lea su versión más actualizada para asegurar consistencia en la ejecución de programas.

Para asegurarla, implementaremos protocolos que serán adoptados por el **controlador de caché**.

Coherencia de caché - *Snooping y Snarfing*

Para adoptar los protocolos señalados, los controladores de caché deben revisar las solicitudes de lectura y escritura que se realizan a sus contrapartes a través de un bus compartido. Este concepto se conoce como ***bus snooping***.

También es posible implementar protocolos con ***snarfing***: cuando un controlador de caché detecta una solicitud de escritura a una dirección de memoria que contiene, actualiza **inmediatamente** el contenido de su caché con los datos de la solicitud. No obstante, esto se utiliza menos por el aumento de latencia en los accesos a memoria y el aumento de tráfico en el bus compartido de datos.

Coherencia de caché - Protocolo MSI

Este protocolo se basa en el uso de estados adicionales al de validez para las líneas de caché, lo que permite detectar errores de coherencia. Su nombre viene de los tres estados posibles:

- ***Modified***: Indica que el contenido de la línea fue modificado y **no es consistente** con el bloque de la memoria principal. Este puede existir **solo en una caché** para un mismo bloque compartido.
- ***Shared***: Indica que el contenido de la línea se encuentra **en al menos una memoria caché**, pero no se ha modificado.
- ***Invalid***: Indica que el contenido de la línea es inválido, que puede ser por dato no existente o por solicitud desde el bus compartido.

Coherencia de caché - Protocolo MSI

Para entender el protocolo, se definen los siguientes **eventos**:

- **Solicitudes del procesador actual**
  - `PrRd:` Solicitud de lectura (*Read* - `Rd`).
  - `PrWr:`. Solicitud de escritura (*Write* - `Wr`).
- **Solicitudes del bus compartido**
  - `BusRd:` Solicitud de un bloque de datos **sin** intención de modificarlo.
  - `BusRdX:`. Solicitud de un bloque de datos **con** intención de modificarlo.
  - `Flush:` Transferencia de línea de caché a la memoria principal.

Coherencia de caché - Protocolo MSI

Cambios de estado de línea de caché por eventos del procesador

- Si se hace la lectura de un dato que no está en memoria o que se invalidó por una modificación (estado ***Invalid***), entonces se lee el bloque de memoria y se copia, el estado de la línea de caché pasa a estado ***Shared*** y se emite la señal `BusRd` al bus compartido.

- Si se hace una lectura en estado ***Shared*** o lectura/escritura en estado ***Modified***, no se emite ninguna señal en el bus compartido y se realiza la operación directamente sobre la caché.

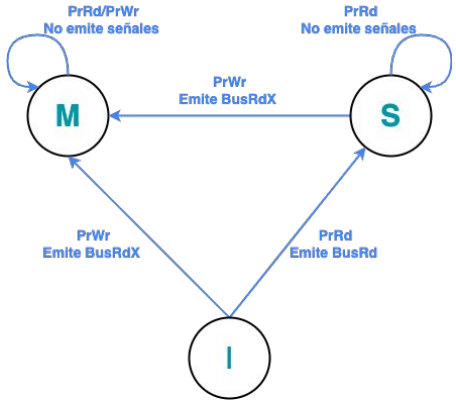


Diagrama de estado de una línea de caché para el protocolo MSI y eventos del procesador.

Coherencia de caché - Protocolo MSI

Cambios de estado de línea de caché por eventos del procesador

- Si se hace una escritura en estado ***Invalid***, se lee el bloque de memoria y se copia, se modifica el contenido directamente en caché, el estado de la línea pasa a estado ***Modified*** y se emite la señal `BusRdX` al bus compartido.

- Si se hace una escritura en estado ***Shared***, se realizan las mismas acciones pero se evita la copia del bloque de memoria al ya encontrarse en caché.

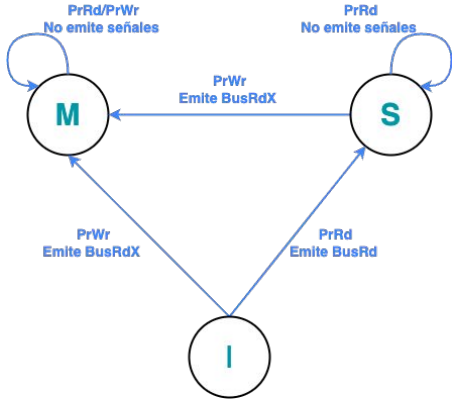


Diagrama de estado de una línea de caché para el protocolo MSI y eventos del procesador.

Coherencia de caché - Protocolo MSI

Cambios de estado de línea de caché por eventos del bus

- Si un controlador recibe la señal `BusRd` o `BusRdX` para una de sus líneas en estado ***Invalid***, mantiene su estado sin hacer nada. Lo mismo ocurre para la señal `BusRd` y una línea en estado ***Shared***.

- Si un controlador recibe la señal `BusRdX` para una de sus líneas en estado ***Shared***, significa que su contenido dejará de ser válido porque el contenido se modificará, por lo que se actualiza a estado ***Invalid***.

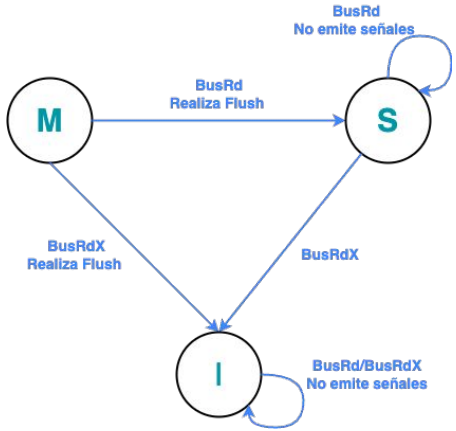


Diagrama de estado de una línea de caché para el protocolo MSI y eventos del bus compartido.

Coherencia de caché - Protocolo MSI

Cambios de estado de línea de caché por eventos del bus

- Si un controlador recibe la señal `BusRd` para una de sus líneas en estado ***Modified***, actualiza su estado a ***Shared*** y realiza `Flush` (escritura de la línea en la memoria principal) para que el controlador que emitió la señal pueda hacer la lectura correcta de memoria.

- Si un controlador recibe la señal `BusRdX` para una de sus líneas en estado ***Modified***, también realiza `Flush` pero pasa a estado ***Invalid*** ya que el controlador que emitió la señal será el único habilitado para poseer la línea modificada.

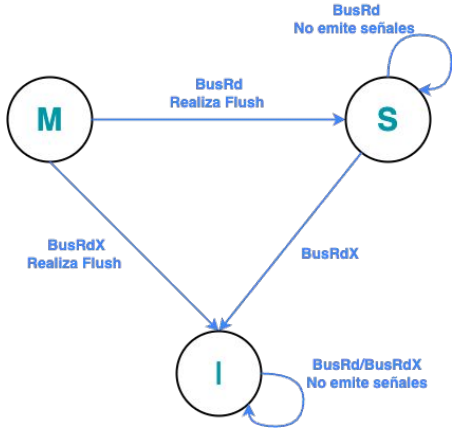


Diagrama de estado de una línea de caché para el protocolo MSI y eventos del bus compartido.

Coherencia de caché - Protocolo MSI

Diagrama de estado general

- Incluyendo ambos tipos de cambio, así se ve el diagrama de estado de una línea de caché para el protocolo MSI.

- Algunas implementaciones permiten que un bloque que ya se encuentre compartido pueda ser transferido por el bus para evitar una lectura de la memoria principal (evento `BusUpgr`).

- **Problema del protocolo:** Usamos el estado ***Shared*** aunque solo una caché posea el bloque.

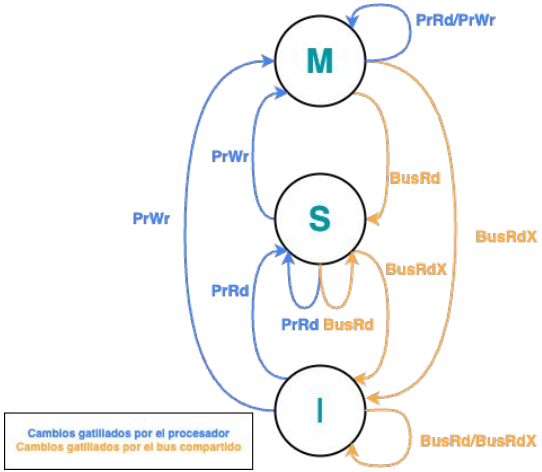


Diagrama de estado de una línea de caché para el protocolo MSI.

Coherencia de caché - Protocolo MESI

Este protocolo es una mejora sobre MSI, desarrollado en la Universidad de Illinois (y, por ende, también conocido como “Protocolo Illinois”). En este protocolo, el estado ***Shared*** cambia su uso y se agrega otro nuevo:

- ***Shared*: Indica que el contenido de la línea se encuentra en **más de una caché** (al menos dos).**
- ***Exclusive*: Indica que el contenido de la línea se encuentra **exclusivamente en dicha caché**, pero sin modificaciones.**

Coherencia de caché - Protocolo MESI

Además de los eventos anteriores, se añade la confirmación (o no) de que el contenido de una línea de caché exista o no en otra:

- `BusRd(S)`: Indica que se realiza una lectura y se obtuvo una confirmación **afirmativa** de la existencia del contenido en otra caché.

- `BusRd(¬S)`: Indica que se realiza una lectura y se obtuvo una confirmación **negativa** de la existencia del contenido en otra caché (por lo que sabemos que solo existirá en la caché actual).

Coherencia de caché - Protocolo MESI

Cambios de estado de línea de caché por eventos del procesador

- La mayoría de las transiciones queda igual, así que veremos las que involucran al nuevo estado.

- Si se hace la lectura de un dato que no está en memoria y que se confirma que no está en ninguna otra caché, el estado de la línea pasa a estado ***Exclusive***. Si se realiza una lectura en este estado, se mantiene igual y no se emiten señales. Si se realiza una escritura, pasa a estado ***Modified*** pero no envía ninguna señal al bus compartido.

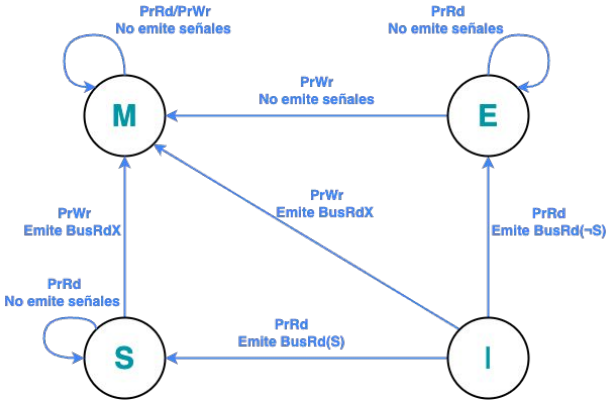


Diagrama de estado de una línea de caché para el protocolo MESI y eventos del bus compartido.

Coherencia de caché - Protocolo MESI

Cambios de estado de línea de caché por eventos del bus

- La mayoría de las transiciones queda igual, así que veremos las que involucran al nuevo estado.

- Si un controlador recibe la señal `BusRd` para una de sus líneas en estado ***Exclusive***, entonces esta ya no es la única y por ende cambia a estado ***Shared***. Si en cambio recibe la señal `BusRdX`, pasa a estado ***Invalid*** ya que no solo no será la única caché que tiene el contenido, sino que no tendrá la última versión de este.

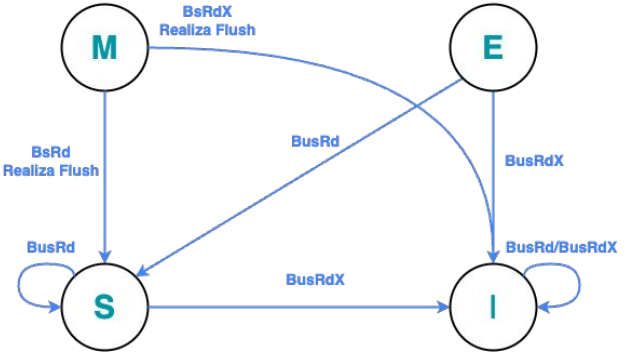


Diagrama de estado de una línea de caché para el protocolo MESI y eventos del procesador.

Coherencia de caché - Protocolo MESI

Diagrama de estado general

- Incluyendo ambos tipos de cambio, así se ve el diagrama de estado de una línea de caché para el protocolo MESI.

- El hecho de no gatillar la señal `BusRdX` para todo caso de escritura implica una ganancia importante de eficiencia respecto al tráfico de bus compartido y revisiones que deben hacer los controladores.

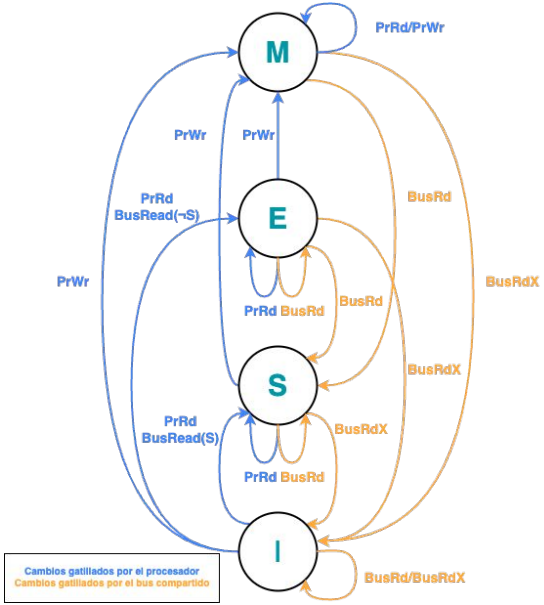


Diagrama de estado de una línea de caché para el protocolo MESI.

Coherencia de caché - Protocolo MESI

Ejercicio en clases

Considere la siguiente memoria compartida con los datos señalados en la tabla.

Dirección	*Label*	Valor
0xCD	i	5
0xCE	temp	-45
0xCF	var1	2
0xD0	var2	9
0xD1	arr	20
0xD2		-10
0xD3		3
0xD4		9

Esta es compartida por tres CPUs que ejecutan simultáneamente los siguientes programas:

CPU0

```
MOV A,(var1)
MOV B,(var2)
ADD A,B
MOV (arr),A
ADD A,(i)
MOV (temp),A
```

CPU1

```
MOV A,Arr
ADD A,1
MOV (temp),A
MOV B,(temp)
MOV B,(B)
MOV (var1),B
```

CPU2

```
MOV B,(i)
MOV A,(arr)
ADD A,B
MOV (i),A
INC B
MOV (var2),B
```

Coherencia de caché - Protocolo MESI

Ejercicio en clases

Indique los valores finales de las variables para cada caché privada y complete las siguientes tablas según cómo se actualiza cada celda a partir del protocolo MESI (indicando con una letra qué bit está activo). Puede asumir que cada variable se almacena en una línea de la caché.

	CPU0							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1								
2								
3								
4								
5								
6								

	CPU1							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1								
2								
3								
4								
5								
6								

	CPU2							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1								
2								
3								
4								
5								
6								

Coherencia de caché - Protocolo MESI

Ejercicio en clases

Ejecución de instrucción 1

- CPU0: `MOV A,(var1)`
- CPU1: `MOV A,arr`
- CPU2: `MOV B,(i)`

CPU0

```
i = -
temp = -
var1 = 2
var2 = -
arr[0] = -
arr[1] = -
A = 2
B = -
```

CPU1

```
i = -
temp = -
var1 = -
var2 = -
arr[0] = -
arr[1] = -
A = 0xD1
B = -
```

CPU2

```
i = 5
temp = -
var1 = -
var2 = -
arr[0] = -
arr[1] = -
A = -
B = 5
```

	CPU0							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	E	I	I	I	I	I
2								
3								
4								
5								
6								

	CPU1							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	I	I	I	I	I	I
2								
3								
4								
5								
6								

	CPU2							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	E	I	I	I	I	I	I	I
2								
3								
4								
5								
6								

CPU0 y CPU2 parten con una línea exclusiva ya que son los únicos que poseen el dato. CPU1 ejecuta un `MOV A,Lit`, por lo que no lee de memoria.

Coherencia de caché - Protocolo MESI

Ejercicio en clases

Ejecución de instrucción 2

- CPU0: `MOV B,(var2)`
- CPU1: `ADD A,1`
- CPU2: `MOV A,(arr)`

CPU0

```
i = -
temp = -
var1 = 2
var2 = 9
arr[0] = -
arr[1] = -
A = 2
B = 9
```

CPU1

```
i = -
temp = -
var1 = -
var2 = -
arr[0] = -
arr[1] = -
A = 0xD2
B = -
```

CPU2

```
i = 5
temp = -
var1 = -
var2 = -
arr[0] = 20
arr[1] = -
A = 20
B = 5
```

	CPU0							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	E	I	I	I	I	I
2	I	I	E	E	I	I	I	I
3								
4								
5								
6								

	CPU1							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	I	I	I	I	I	I
2	I	I	I	I	I	I	I	I
3								
4								
5								
6								

	CPU2							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	E	I	I	I	I	I	I	I
2	E	I	I	I	E	I	I	I
3								
4								
5								
6								

CPU0 y CPU2 ahora poseen dos líneas exclusivas y CPU1 sigue con todas sus líneas inválidas.

Coherencia de caché - Protocolo MESI

Ejercicio en clases

Ejecución de instrucción 3

- CPU0: `ADD A,B`
- CPU1: `MOV (temp),A`
- CPU2: `ADD A,B`

CPU0

```
i = -
temp = -
var1 = 2
var2 = 9
arr[0] = -
arr[1] = -
A = 11
B = 9
```

CPU1

```
i = -
temp = 0xD2
var1 = -
var2 = -
arr[0] = -
arr[1] = -
A = 0xD2
B = -
```

CPU2

```
i = 5
temp = -
var1 = -
var2 = -
arr[0] = 20
arr[1] = -
A = 25
B = 5
```

	CPU0							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	E	I	I	I	I	I
2	I	I	E	E	I	I	I	I
3	I	I	E	E	I	I	I	I
4								
5								
6								

	CPU1							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	I	I	I	I	I	I
2	I	I	I	I	I	I	I	I
3	I	M	I	I	I	I	I	I
4								
5								
6								

	CPU2							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	E	I	I	I	I	I	I	I
2	E	I	I	I	E	I	I	I
3	E	I	I	I	E	I	I	I
4								
5								
6								

CPU1 almacena directamente la línea de temp en estado M por escribir sobre ella; no se invalidan en CPU0 ni CPU2 ya que no la poseen.

Coherencia de caché - Protocolo MESI

Ejercicio en clases

Ejecución de instrucción 4

- CPU0: `MOV (arr),A`
- CPU1: `MOV B,(temp)`
- CPU2: `MOV (i),A`

CPU0

```
i = -
temp = -
var1 = 2
var2 = 9
arr[0] = 11
arr[1] = -
A = 11
B = 9
```

CPU1

```
i = -
temp = 0xD2
var1 = -
var2 = -
arr[0] = -
arr[1] = -
A = 0xD2
B = 0xD2
```

CPU2

```
i = 25
temp = -
var1 = -
var2 = -
arr[0] = 20
arr[1] = -
A = 25
B = 5
```

	CPU0							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	E	I	I	I	I	I
2	I	I	E	E	I	I	I	I
3	I	I	E	E	I	I	I	I
4	I	I	E	E	M	I	I	I
5								
6								

	CPU1							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	I	I	I	I	I	I
2	I	I	I	I	I	I	I	I
3	I	M	I	I	I	I	I	I
4	I	M	I	I	I	I	I	I
5								
6								

	CPU2							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	E	I	I	I	I	I	I	I
2	E	I	I	I	E	I	I	I
3	E	I	I	I	E	I	I	I
4	M	I	I	I	I	I	I	I
5								
6								

CPU1 lee de su caché una línea que solo posee él, no invalida nada. CPU2 pasa a estado M para la línea de i, no invalida nada. CPU0 modifica arr[0] y almacena en su línea el estado M; invalida el contenido de la caché de CPU2.

Coherencia de caché - Protocolo MESI

Ejercicio en clases

Ejecución de instrucción 5

- CPU0: `ADD A,(i)`
- CPU1: `MOV B,(B)`
- CPU2: `INC B`

CPU0

```
i = 25
temp = -
var1 = 2
var2 = 9
arr[0] = 11
arr[1] = -
A = 36
B = 9
```

CPU1

```
i = -
temp = 0xD2
var1 = -
var2 = -
arr[0] = -
arr[1] = -10
A = 0xD2
B = -10
```

CPU2

```
i = 25
temp = -
var1 = -
var2 = -
arr[0] = 20
arr[1] = -
A = 25
B = 6
```

	CPU0							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	E	I	I	I	I	I
2	I	I	E	E	I	I	I	I
3	I	I	E	E	I	I	I	I
4	I	I	E	E	M	I	I	I
5	S	I	E	E	M	I	I	I
6								

	CPU1							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	I	I	I	I	I	I
2	I	I	I	I	I	I	I	I
3	I	M	I	I	I	I	I	I
4	I	M	I	I	I	I	I	I
5	I	M	I	I	I	E	I	I
6								

	CPU2							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	E	I	I	I	I	I	I	I
2	E	I	I	I	E	I	I	I
3	E	I	I	I	E	I	I	I
4	M	I	I	I	I	I	I	I
5	S	I	I	I	I	I	I	I
6								

CPU1 queda con línea exclusiva de arr[1]. Por la lectura de i de CPU0, su línea pasa a estado S mientras que la línea de CPU2 también pasa a estado S, haciendo *flush* para que CPU0 pueda leer el contenido modificado.

Coherencia de caché - Protocolo MESI

Ejercicio en clases

Ejecución de instrucción 6

- CPU0: `MOV (temp),A`
- CPU1: `MOV (var1),B`
- CPU2: `MOV (var2),B`

CPU0

```
i = 25
temp = 36
var1 = 2
var2 = 9
arr[0] = 11
arr[1] = -
A = 36
B = 9
```

CPU1

```
i = -
temp = 0xD2
var1 = -10
var2 = -
arr[0] = -
arr[1] = -10
A = 0xD2
B = -10
```

CPU2

```
i = 25
temp = -
var1 = -
var2 = 6
arr[0] = 20
arr[1] = -
A = 25
B = 6
```

	CPU0							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	E	I	I	I	I	I
2	I	I	E	E	I	I	I	I
3	I	I	E	E	I	I	I	I
4	I	I	E	E	M	I	I	I
5	S	I	E	E	M	I	I	I
6	S	M	I	I	M	I	I	I

	CPU1							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	I	I	I	I	I	I	I	I
2	I	I	I	I	I	I	I	I
3	I	M	I	I	I	I	I	I
4	I	M	I	I	I	I	I	I
5	I	M	I	I	I	E	I	I
6	I	I	M	I	I	E	I	I

	CPU2							
Instrucción	i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
1	E	I	I	I	I	I	I	I
2	E	I	I	I	E	I	I	I
3	E	I	I	I	E	I	I	I
4	M	I	I	I	I	I	I	I
5	S	I	I	I	I	I	I	I
6	S	I	I	M	I	I	I	I

CPU0 queda con la línea temp modificada y se invalida en CPU1. CPU1 queda con la línea var1 modificada y se invalida en CPU0. CPU2 queda con la línea var2 modificada y se invalida en CPU0.

Coherencia de caché - Protocolo MESI

Ejercicio en clases

Estado final

Se destaca que las variables **rojas** señalan que la línea quedó con el contenido de estas, pero fue invalidado por modificaciones de otras caché.

CPU0	CPU1	CPU2
i = 25 temp = 36 var1 = 2 var2 = 9 arr[0] = 11 arr[1] = - A = 36 B = 9	i = - temp = 0xD2 var1 = -10 var2 = - arr[0] = - arr[1] = -10 A = 0xD2 B = -10	i = 25 temp = - var1 = - var2 = 6 arr[0] = 20 arr[1] = - A = 25 B = 6

CPU0

i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
S	M	I	I	M	I	I	I

CPU1

i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
I	I	M	I	I	E	I	I

CPU2

i	temp	var1	var2	arr[0]	arr[1]	arr[2]	arr[3]
S	I	I	M	I	I	I	I

Notar que no se incluyen `arr[2]` ni `arr[3]` porque nunca se leen ni modifican.

Coherencia de caché - Transferencia entre cachés

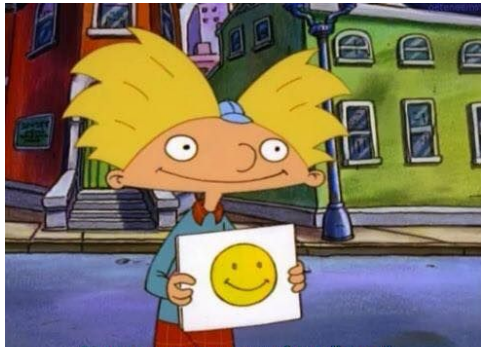
Una mejora que se puede hacer en los protocolos es habilitar la transferencia de datos entre cachés. Así, si un controlador solicita un bloque que ya se encuentra contenido en una caché, su contenido se obtiene a través del bus compartido y no de la memoria principal.

Esto permite nuevos protocolos, por ejemplo, **MOESI**, donde se agrega el estado ***Ownership*** para señalar que una caché es "dueña" del contenido de una línea y es la única con permisos para modificarla. Si otra caché la necesita, esta se encarga de transmitir los datos. Si modifica su contenido, puede transmitirlo al resto de las cachés para mantener consistencia y ahorrar escrituras en memoria.

Coherencia de caché

Existen más mejoras y variantes de protocolos que se pueden implementar, por ejemplo: MOSI, MESIF, MOESIF.

No obstante, estos ya son parte de contenidos más avanzados y contenidos, por lo que retomaremos el contenido de las otras variantes de arquitecturas paralelas.

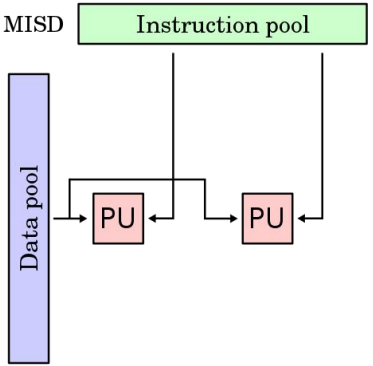


Tipos de arquitectura paralela - MISD

Multiple Instruction streams, Single Data stream

Múltiples flujos de instrucciones operan sobre un único flujo de datos en múltiples unidades de programación (PU).

Esta categoría es más bien teórica ya que existen **muy** pocos ejemplos prácticos donde su uso puede ser relevante, siendo estos muy especializados.



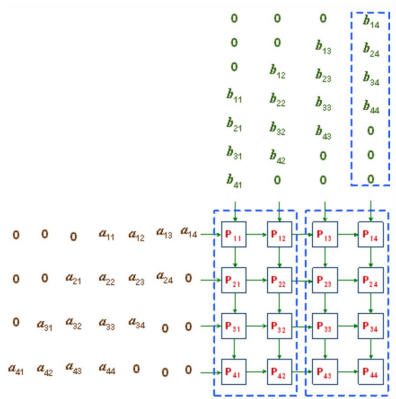
Tipos de arquitectura paralela - MISD

Arreglos sistólicos: Su nombre nace por su analogía con la presión arterial sistólica: Un flujo de datos se transmite por distintos procesadores que realizan, de forma sincronizada, distintas operaciones sobre ellos.

Ejemplo 1: Multiplicación matricial.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

$$\begin{aligned}
c_{11} &= a_{11} \bullet b_{11} + a_{12} \bullet b_{21} + a_{13} \bullet b_{31} + a_{14} \bullet b_{41} \\
c_{12} &= a_{11} \bullet b_{12} + a_{12} \bullet b_{22} + a_{13} \bullet b_{32} + a_{14} \bullet b_{42} \\
c_{13} &= a_{11} \bullet b_{13} + a_{12} \bullet b_{23} + a_{13} \bullet b_{33} + a_{14} \bullet b_{43} \\
c_{14} &= a_{11} \bullet b_{14} + a_{12} \bullet b_{24} + a_{13} \bullet b_{34} + a_{14} \bullet b_{44} \\
c_{21} &= a_{21} \bullet b_{11} + a_{22} \bullet b_{21} + a_{23} \bullet b_{31} + a_{24} \bullet b_{41} \\
c_{22} &= a_{21} \bullet b_{12} + a_{22} \bullet b_{22} + a_{23} \bullet b_{32} + a_{24} \bullet b_{42} \\
c_{23} &= a_{21} \bullet b_{13} + a_{22} \bullet b_{23} + a_{23} \bullet b_{33} + a_{24} \bullet b_{43} \\
c_{24} &= a_{21} \bullet b_{14} + a_{22} \bullet b_{24} + a_{23} \bullet b_{34} + a_{24} \bullet b_{44} \\
c_{31} &= a_{31} \bullet b_{11} + a_{32} \bullet b_{21} + a_{33} \bullet b_{31} + a_{34} \bullet b_{41} \\
c_{32} &= a_{31} \bullet b_{12} + a_{32} \bullet b_{22} + a_{33} \bullet b_{32} + a_{34} \bullet b_{42} \\
c_{33} &= a_{31} \bullet b_{13} + a_{32} \bullet b_{23} + a_{33} \bullet b_{33} + a_{34} \bullet b_{43} \\
c_{34} &= a_{31} \bullet b_{14} + a_{32} \bullet b_{24} + a_{33} \bullet b_{34} + a_{34} \bullet b_{44} \\
c_{41} &= a_{41} \bullet b_{11} + a_{42} \bullet b_{21} + a_{43} \bullet b_{31} + a_{44} \bullet b_{41} \\
c_{42} &= a_{41} \bullet b_{12} + a_{42} \bullet b_{22} + a_{43} \bullet b_{32} + a_{44} \bullet b_{42} \\
c_{43} &= a_{41} \bullet b_{13} + a_{42} \bullet b_{23} + a_{43} \bullet b_{33} + a_{44} \bullet b_{43} \\
c_{44} &= a_{41} \bullet b_{14} + a_{42} \bullet b_{24} + a_{43} \bullet b_{34} + a_{44} \bullet b_{44}
\end{aligned}$$



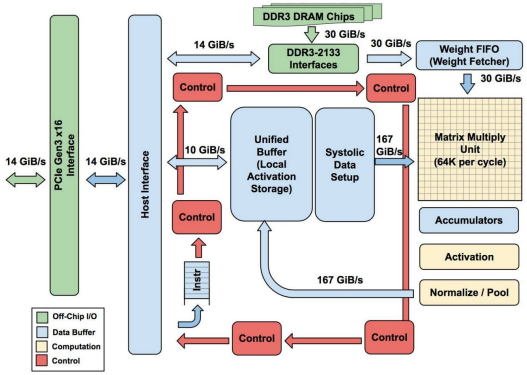
En esta arquitectura, cada unidad $P_{i,j}$ posee un registro que almacena el valor $c_{i,j}$ y realiza la siguiente operación:

1. Calcula el producto de las entradas *a*, *b*
2. Guarda en el registro $c_{i,j}$ la suma del estado anterior y el producto calculado, *i.e.* $c_{i,j}$ += *a* * *b*
3. Transmite el valor de entrada "*a*" a la unidad $P_{i,j+1}$
4. Transmite el valor de entrada "*b*" a la unidad $P_{i+1,j}$

Finalizada la ejecución, cada unidad $P_{i,j}$ poseerá el valor $c_{i,j}$ final de la multiplicación matricial (fuente).

Tipos de arquitectura paralela - MISD

Ejemplo 2: *Tensor Processing Unit* (TPU) de Google. Unidad creada con foco en arquitecturas específicas para un dominio: *Machine Learning*. Estas calculan multiplicaciones matriciales (por ejemplo, para convoluciones en redes neuronales) en tiempos sustancialmente menores con el uso de arreglos sistólicos como se indicó en el ejemplo anterior.



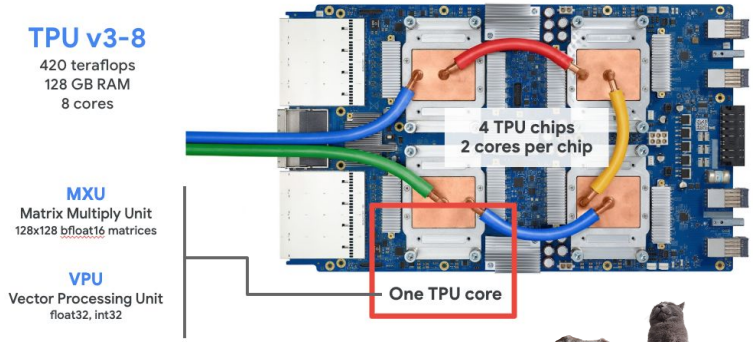


Diagrama general y chip con 8 *cores* para la arquitectura de una TPU. Para más información:

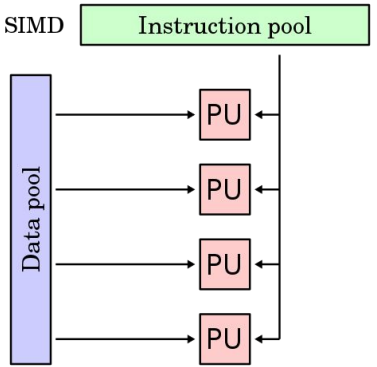
- Fuente 1
- Fuente 2
- Fuente 3
- Fuente 4
- Fuente 5

Tipos de arquitectura paralela - SIMD

Single Instruction stream, Multiple Data streams

Un único flujo de instrucciones opera sobre múltiples flujos de datos en múltiples unidades de programación (PU).

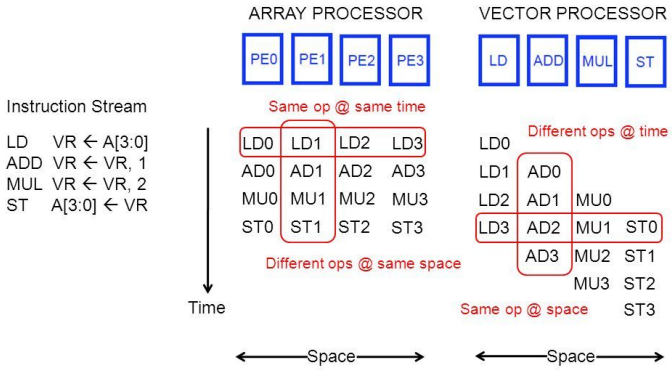
Este tipo de problema se da de forma natural en muchas aplicaciones: cálculos vectoriales y matriciales; transformación de datos en procesos ETL (*Extract*, *Transform*, *Load*); procesamiento de imágenes; *Machine Learning*; entre muchas más.



Tipos de arquitectura paralela - SIMD

Los tipos de arquitectura SIMD explotan el paralelismo a nivel de datos: la aplicación de **un mismo programa** o conjunto de instrucciones **a distintos fragmentos de datos**. Se pueden clasificar de la siguiente forma:

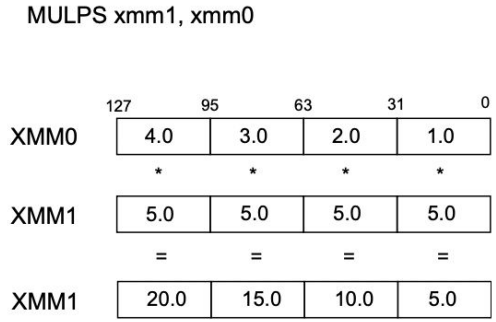
- ***Array Processor***: Una instrucción opera sobre múltiples datos **al mismo tiempo** usando **múltiples PU's**.
- ***Vector Processor***: Una instrucción opera sobre múltiples datos **en iteraciones consecutivas** en una **única PU** (*pipeline*).



Ejemplo de ejecución de un mismo programa en un *array processor* vs. un *vector processor*.

Tipos de arquitectura paralela - SIMD

Ejemplo 1: Instrucciones multimedia. Intel ha añadido extensiones a su ISA x86 enfocadas en mejorar los tiempos de ejecución de programas multimedia (*encoding* de videos, audio, etc.). En 1999 crean la extensión **SSE** (***S****treaming* **S**IMD ***E****xtension*), que agrega una unidad de ejecución adicional al procesador y 8 registros de 128 bits cada uno (`XMM0-XMM7`) con la capacidad de almacenar más de un número para ser operados de forma simultánea.



Ejemplo de la instrucción `MULPS` de SSE, que permite realizar la multiplicación simultánea de 4 números de 32 bits entre dos registros `XMM`.

Tipos de arquitectura paralela - SIMD

Ejemplo 2: GPU (*G*raphics *P*rocessing *U*nit).

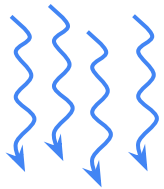
Dispositivo I/O especializado en problemas “ridículamente paralelos”. La CPU se comunica con esta unidad para traspasar la ejecución de tareas o programas. La CPU es ideal para el **paralelismo de tareas** y la GPU para el **paralelismo de datos**.

La GPU se clasifica como **SIMT** (*S*ingle *I*nstruction, *M*ultiple *T*hreads), ya que ejecuta múltiples ***threads*** concurrentemente sobre un mismo programa, pero con datos distintos.

Tipos de arquitectura paralela - SIMD

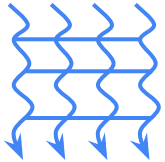
Comparativa de arquitecturas

MIMD: Múltiples *threads* independientes



SIMD: Un *thread* con gran amplitud de datos

SIMT: Múltiples *threads* sincronizados (*warp*)



Arquitectura	CPU Multicore (MIMD)	x86 SSE (SIMD)	GPU (SIMT)
Ventaja	Buen rendimiento para tareas de propósito general	Permite mezclar código secuencial y código paralelo	Lectura y escritura más eficiente (por bloque para todos los *threads*)
Desventaja	Mal funcionamiento para paralelismo de datos	Aumento de complejidad para lectura/escritura de memoria	Latencia de memoria (por esperar a todos los *threads*); divergencia de código (*threads* ejecutando distintas instrucciones por control de flujo)

Tipos de arquitectura paralela - SIMD

Ejemplo de GPU: GeForce GTX 580

- Ejecuta “simultáneamente” 48 *warps* de 32 *threads* cada uno, separados en dos grupos de 16 (*dual warp scheduler*).
- Posee 16 SMs (*Streaming Multiprocessor*) de 32 *cores* cada uno, permitiendo la ejecución concurrente de 2 *warps* 16 *threads* (1 *thread* por *core*).
- Puede almacenar la ejecución de hasta **24576 *threads*** (16 * 48 * 32) y posee una *performance* de **1580 GFLOPS/s** (1GFLOP/s = 1 billón de operaciones de punto flotante por segundo).

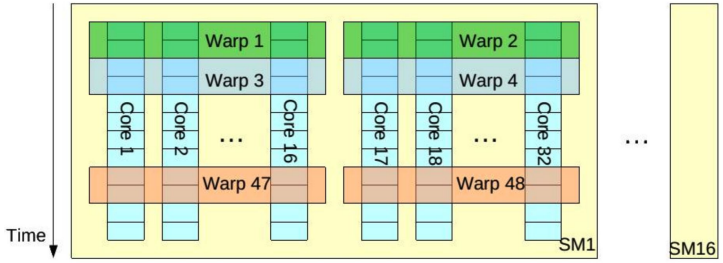


Imagen de la GPU GTX 580 y un diagrama general de su composición, así como la ejecución de los *warps* en el tiempo. Información de la evolución de este tipo de arquitectura de NVIDIA aquí.

Tipos de arquitectura paralela

Para finalizar, actualmente se ocupan dos categorizaciones nuevas que van en línea con cómo se utilizan los procesadores en la actualidad:

- **SPMD: *Single Program, Multiple Data Streams***. Múltiples procesadores ejecutan simultáneamente un mismo programa.
- **MPMD: *Multiple Programs*, *Multiple Data Streams***. Múltiples programas se ejecutan simultáneamente en múltiples procesadores.

¿Ejercicios?

Para este contenido las preguntas suelen ser más bien conceptuales, por lo que no hay nada que ejercitar para esta ocasión.

Con esto, **damos por finalizada la materia del curso**.

Antes de terminar

¿Dudas?

¿Consultas?

¿Inquietudes?

¿Comentarios?

DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343

Arquitectura de Computadores

Clase 14 - Paralelismo Avanzado y Coherencia de Caché

Profesor: Germán Leandro Contreras Sagredo