中山大學
SUN YAT-SEN UNIVERSITY

国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

Computer Architecture

计 算 机 体 系 结 构

第16讲：TLP（2）

张献伟

xianweiz.github.io

DCS3013, 11/28/2022

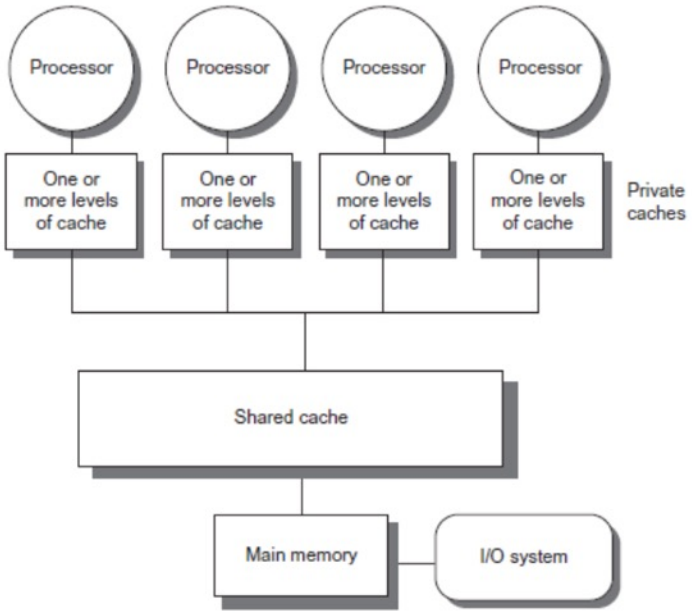
Quiz Questions

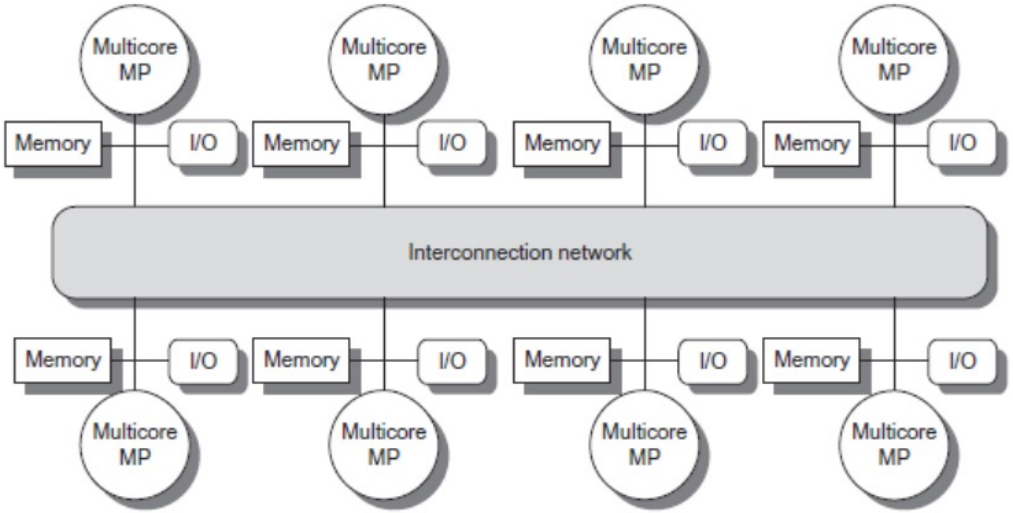
For remote attendees, plz email to zhangxw79@mail.sysu.edu.cn (ddl: 14:40).

- Q1: a cache: capacity is 16KB, 4-way associative, block is 32B. Split the 32b address into tag, index, offset.
  #sets = 16KB/32B/4 = 128, [31-12][11-5][4-0]
- Q2: for the above 16KB cache. How can you improve its performance? List 2 techniques.
  Higher associativity; critical word first; way prediction; victim …
- Q3: DRAM interface is 64b, chip is 4b wide and 4Gb, what's the rank capacity?
  (64b/4b) * 4Gb = 64Gb = 8GB
- Q4: why HBM is of much higher bw than DDR/GDDR?
  Much wider interface (stacking/closer to processor).
- Q5: list 3 advantages of NVM, compared to DRAM.
  Non-volatile, higher capacity, lower cost, …

Shared Memory[共享内存]

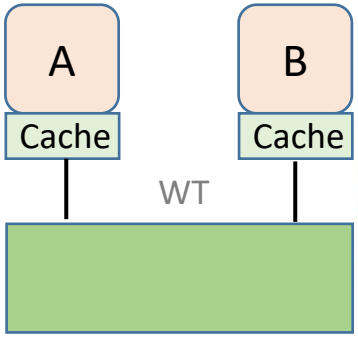
- The term "**shared memory**" associated with both SMP and DSM refers to the fact that the address space is shared
  - Communication among threads occurs through the shared address space
  - Thus, a memory reference can be made by any processor to any memory location





Cache Coherence[缓存一致性]

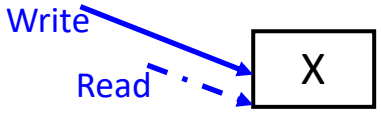
- Processors may see different values of the same data
  - The view of memory held by two different processors is through their individual caches, which, without any additional precautions, could end up seeing two different values
- Cache coherence problem[缓存一致性问题]
  - Conflicts between global state (main memory) and local state (private cache)
  - At time 4, what if processor B reads X?



Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

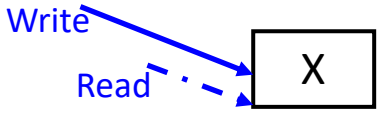
A memory system is coherent, if

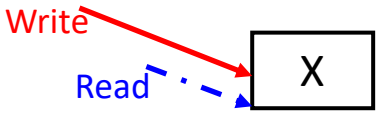
- A read by processor P to location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always return the value written by P
  - Preserves program order ①



A memory system is coherent, if

- A read by processor P to location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always return the value written by P
  - Preserves program order ①

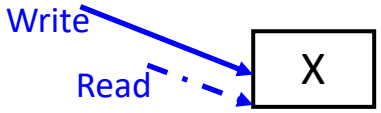


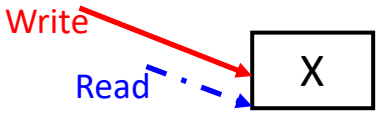


- A read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses
  - Defines the notion of what it means to have a coherent view of memory ②

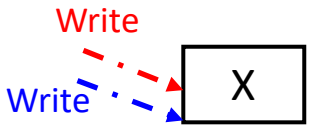
A memory system is coherent, if

- A read by processor P to location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always return the value written by P
  - Preserves program order ①



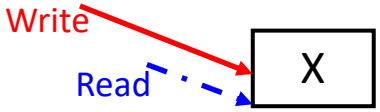


- A read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses
  - Defines the notion of what it means to have a coherent view of memory ②
- Writes to the same location are serialized; that is, two writes to the same location by any two processors are seen in the same order by all processors
  - Write serialization ③



Consistency also Matters[内存一致性]

- The three properties ①②③ are sufficient to ensure coherence
- However, when a written value will be seen is also important
  - A write of X on one processor precedes a read of X on another processor by a very small time, it may be impossible to ensure that the read returns the value of the data written, since the written data may not even have left the processor at that point



- **Memory consistency**: when a written value must been seen by a reader

Thread 1

```
(1) A = 1
(2) print(B)
```

Thread 2

```
(3) B = 1
(4) print(A)
```

A and B are initially both 0

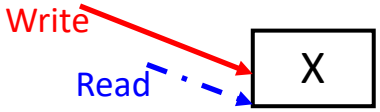
What this program can output?

- 01: (1)(2)(3)(4) or (3)(4)(1)(2)
- 11: (1)(3)(2)(4) or (1)(3)(4)(2)
- 00?

https://www.cs.utexas.edu/~bornholt/post/memory-models.html

Consistency also Matters[内存一致性]

- The three properties ①②③ are sufficient to ensure coherence
- However, when a written value will be seen is also important
  - A write of X on one processor precedes a read of X on another processor by a very small time, it may be impossible to ensure that the read returns the value of the data written, since the written data may not even have left the processor at that point



- **Memory consistency**: when a written value must been seen by a reader

Thread 1	Thread 2
(1) A = 1	(3) B = 1
(2) print(B)	(4) print(A)

A and B are initially both 0

What this program can output?

- 01: (1)(2)(3)(4) or (3)(4)(1)(2)
- 11: (1)(3)(2)(4) or (1)(3)(4)(2)
- 00?

A=0: (4)->(1)
B=0: (2)->(3)

(4)->(1)->(2)->(3) ➔ (4)

SC

https://www.cs.utexas.edu/~bornholt/post/memory-models.html

Coherence vs. Consistency[对比]

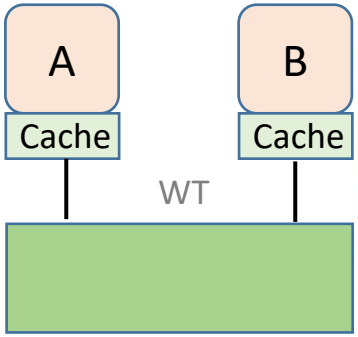
- Coherence[缓存一致性]
  - Defines what values can be returned by a read
  - All reads by any processor must return the most recently written value
  - Writes to the **same location** by any two processors are seen in the same order by all processors
- Consistency[内存一致性]
  - Determines when a written value will be returned by a read
  - Consistency insures that writes to **different locations** will be seen in an order that *makes sense*, given the source code
  - If a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

Enforcing Coherence[保证一致性]

- Coherent caches provide
  - **Migration**: movement of data[搬运]
    - A data item can be moved to a local cache and used there in a transparent fashion
  - **Replication**: multiple copies of data[备份]
    - Make a copy of the data item in the local cache, so that shared data can be simultaneously read
- Whose responsibility? Software?
  - Can programmer ensure coherence if caches invisible to sw?
  - What if the ISA provided a cache flush instruction?
    - FLUSH-LOCAL A: flushes/invalidates the cache block containing address A from a processor's local cache
    - FLUSH-GLOBAL A: flushes/invalidates the cache block containing address A from all other processors' caches
    - FLUSH-CACHE X: flushes/invalidates all blocks in cache X

Enforcing Coherence (cont.)

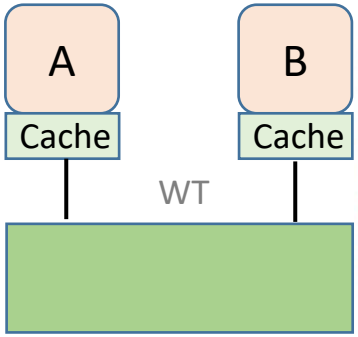
- Software solutions are of high overheads
  - And, programming burden
- Multiprocessors adopt a hardware solution to maintain coherent caches[硬件方案]
  - Supporting the migration and replication is critical to performance in accessing shared data
- For the example,
  - Invalidate all other copies of X when A writes to it



Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

Enforcing Coherence (cont.)

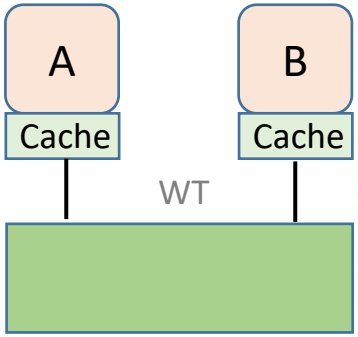
- Software solutions are of high overheads
  - And, programming burden
- Multiprocessors adopt a hardware solution to maintain coherent caches[硬件方案]
  - Supporting the migration and replication is critical to performance in accessing shared data
- For the example,
  - Invalidate all other copies of X when A writes to it



Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

Enforcing Coherence (cont.)

- Software solutions are of high overheads
  - And, programming burden
- Multiprocessors adopt a hardware solution to maintain coherent caches[硬件方案]
  - Supporting the migration and replication is critical to performance in accessing shared data
- For the example,
  - Invalidate all other copies of X when A writes to it

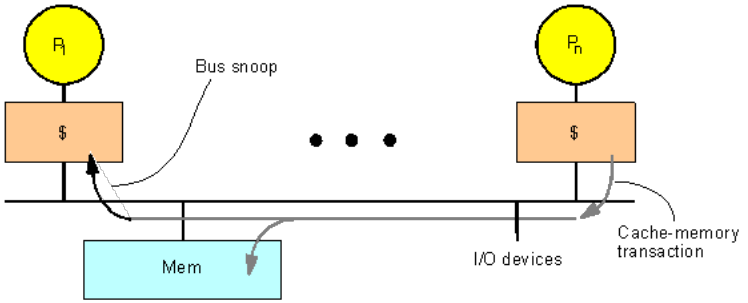


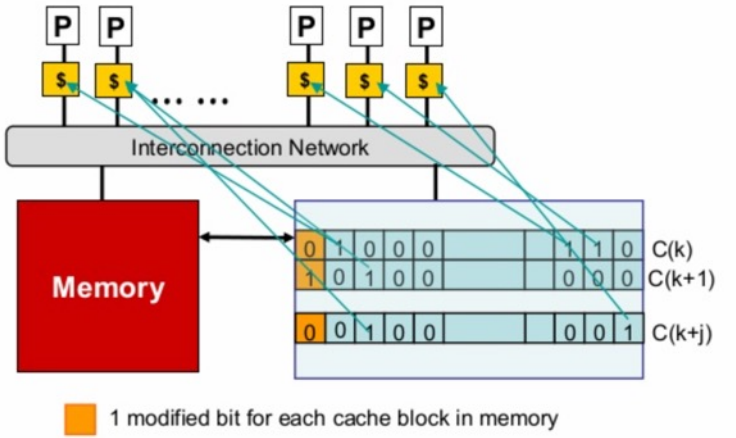
Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1 ✗	0

How do you know which copies to invalidate?

Coherence Protocols[缓存一致性协议]

- Cache **coherence protocols**: the rules to maintain coherence for multiple processors
  - Key is to track the state of any sharing of a data block
- Two classes of protocols
  - Snooping[窥探]
    - Each core tracks sharing status of each block
  - Directory based[基于目录]
    - Sharing status of each block kept in one location





Snooping Coherence Protocols[窥探]

- Write invalidation protocol[写无效]
  - Ensure that a processor has exclusive access to a data item before it writes that item
  - Exclusive access ensures that no other readable or writable copies of an item exist when the write occurs
    - All other cached copies of the item are **invalidated** (👉 that's the name)
- Write update/broadcast protocol[写更新]
  - Update all the cached copies of data item when that item is written
  - Must broadcast all writes to shared cache lines, and thus consumes considerably more bandwidth
- Write invalidation protocol is by far the most common
  - We'll focus on it

Write Invalidation Protocol[写无效]

- Write invalidate
  - On write, invalidate all other copies
  - Use bus itself to serialize
- Example
  - Invalidation protocol working on a snooping bus for a single block (X) with write-back caches

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
				0
Processor A reads X	Cache miss for X	0		0
Processor B reads X	Cache miss for X	0	0	0
Processor A writes a 1 to X	Invalidation for X	1		0
Processor B reads X	Cache miss for X	1	1	1

Write Invalidation Protocol[写无效]

- Write invalidate
  - On write, invalidate all other copies
  - Use bus itself to serialize
- Example
  - Invalidation protocol working on a snooping bus for a single block (X) with write-back caches

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
Neither cache initially holds X and the value of X in memory is 0				0
Processor A reads X	Cache miss for X	0		0
Processor B reads X	Cache miss for X	0	0	0
Processor A writes a 1 to X	Invalidation for X	1		0
Processor B reads X	Cache miss for X	1	1	1

Write Invalidation Protocol[写无效]

- Write invalidate
  - On write, invalidate all other copies
  - Use bus itself to serialize
- Example
  - Invalidation protocol working on a snooping bus for a single block (X) with write-back caches

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
		Neither cache initially holds X and the value of X in memory is 0		0
Processor A reads X	Cache miss for X	0		0
		Processor A reads X, migrating from memory to the local cache		
Processor B reads X	Cache miss for X	0	0	0
Processor A writes a 1 to X	Invalidation for X	1		0
Processor B reads X	Cache miss for X	1	1	1

Write Invalidation Protocol[写无效]

- Write invalidate
  - On write, invalidate all other copies
  - Use bus itself to serialize
- Example
  - Invalidation protocol working on a snooping bus for a single block (X) with write-back caches

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
		Neither cache initially holds X and the value of X in memory is 0		0
Processor A reads X	Cache miss for X	0		0
		Processor A reads X, migrating from memory to the local cache		
Processor B reads X	Cache miss for X	0	0	0
		Processor B reads X, migrating from memory to the local cache		
Processor A writes a 1 to X	Invalidation for X	1		0
Processor B reads X	Cache miss for X	1	1	1

Write Invalidation Protocol[写无效]

- Write invalidate
  - On write, invalidate all other copies
  - Use bus itself to serialize
- Example
  - Invalidation protocol working on a snooping bus for a single block (X) with write-back caches

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
		Neither cache initially holds X and the value of X in memory is 0		0
Processor A reads X	Cache miss for X	0 Processor A reads X, migrating from memory to the local cache		0
Processor B reads X	Cache miss for X	0 Processor B reads X, migrating from memory to the local cache	0	0
Processor A writes a 1 to X	Invalidation for X	1 Processor A writes X, invalidating the copy on B		0
Processor B reads X	Cache miss for X	1	1	1

Write Invalidation Protocol[写无效]

- Write invalidate
  - On write, invalidate all other copies
  - Use bus itself to serialize
- Example
  - Invalidation protocol working on a snooping bus for a single block (X) with write-back caches

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
		Neither cache initially holds X and the value of X in memory is 0		0
Processor A reads X	Cache miss for X	0		0
		Processor A reads X, migrating from memory to the local cache		
Processor B reads X	Cache miss for X	0	0	0
		Processor B reads X, migrating from memory to the local cache		
Processor A writes a 1 to X	Invalidation for X	1		0
		Processor A writes X, invalidating the copy on B		
Processor B reads X	Cache miss for X	1	1	1
		Processor B reads X, A responds with the value canceling the mem response and updates both B's cache and memory		