Chapter 5

Multiprocessors and Thread-Level Parallelism

Soontae Kim

School of Computing, KAIST

Announcements

- Final project presentations
  - June 7
    - Teams with three members and two members
  - June 9
    - Teams with two members and one member
  - Presentation times allowed
    - One-member team: 6 min. Two-member teams: 8 min. Three-member team: 10 min.
  - Contents
    - Motivation, approach, implementation, experimental methodology, experimental results, lessons & conclusion
  - Evaluation criteria
    - Time, correct implementation, results, presentation quality

Announcements

- Final reports are due on June 16
  - Single column, 11 font size, 5~8 pages
  - Follow typical conference format
    - Introduction, related work, approach(proposed idea), experimental environment and results, conclusion
  - Also show the roles of members
- Final exam
  - From 1:00pm to 3:00pm on June 14
  - Cover all class materials after midterm exam

Multiprocessors: major factors

- Growth in data-intensive applications
    - Data bases, Internet, …
- Growing interest in servers, server perf.
- Increasing desktop perf. Is less important
    - Except for graphics
- Improved understanding in how to use multiprocessors effectively
    - Especially server where significant natural TLP exists
- Advantage of leveraging design investment by replication rather than unique design

4

Introduction

- Thread-Level parallelism
  - Have multiple program counters
  - Uses MIMD model
  - TLP from embedded to high-end servers
  - Targeted for tightly-coupled shared-memory multiprocessors controlled by a single OS
  - Two software models
    - Parallel processing: tightly coupled set of threads collaborating on a single task
    - Multiprogramming: multiple relatively independent processes from one or more users

Types

- ## Symmetric multiprocessors (SMP)
  - ### Small number of cores
  - ### Share single memory with uniform memory latency

- ## Distributed shared memory (DSM)
  - ### Memory distributed among processors
  - ### Non-uniform memory access/latency (NUMA)
  - ### Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks
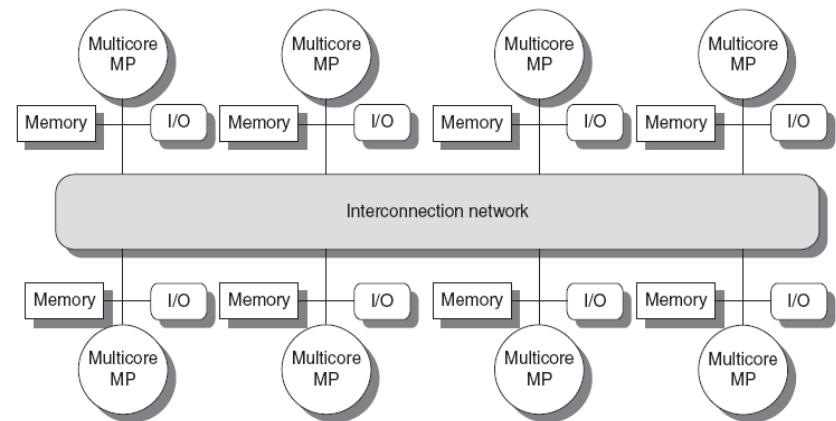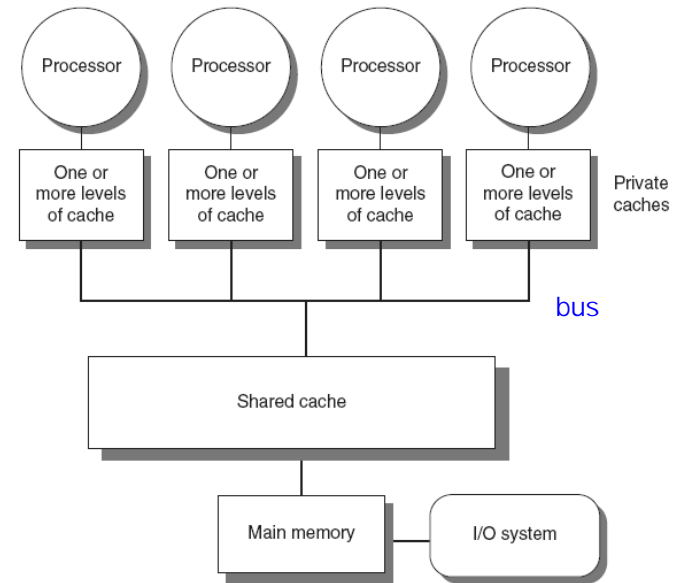
Challenges of Parallel Processing

- First challenge is % of program inherently sequential
- Suppose 80X speedup from 100 processors. What fraction of original program can be sequential?

$$\text{Speedup}_{overall} = \frac{1}{\left(1 - \text{Fraction}_{parallel}\right) + \dfrac{\text{Fraction}_{parallel}}{\text{Speedup}_{parallel}}}$$

$$80 = \frac{1}{\left(1 - \text{Fraction}_{parallel}\right) + \dfrac{\text{Fraction}_{parallel}}{100}}$$

$$80 \times \left(\left(1 - \text{Fraction}_{parallel}\right) + \frac{\text{Fraction}_{parallel}}{100}\right) = 1$$

$$79 = 80 \times \text{Fraction}_{parallel} - 0.8 \times \text{Fraction}_{parallel}$$

$$\text{Fraction}_{parallel} = 79/79.2 = 99.75\%$$

Challenges of Parallel Processing II

- Second challenge is long latency to remote memory

- Suppose 32 CPU MP, 3.3GHz, 200 ns remote memory latency, all local accesses hit memory hierarchy and base CPI is 0.5. (Remote access = 200/0.3 = 666 clock cycles.)

- What is performance impact if 0.2% of instructions involve remote access?

- CPI = Base CPI + Remote request rate x Remote request cost = 0.5 + 0.2% x 666 = 0.5 + 1.2 = 1.7

- No communication is 1.7/0.5 or 3.4 times faster than 0.2% of instructions involving local access

8

Challenges of Parallel Processing III

1. Application parallelism $\Rightarrow$ primarily via new algorithms that have better parallel performance
2. Long remote latency impact $\Rightarrow$ both by architecture and by the programmer

- For example, reduce frequency of remote accesses either by
    - Caching shared data (HW)
    - Restructuring the data layout to make more accesses local (SW)

Cache Coherence

- Processors may see different values through their caches:

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

Defining Coherent Memory System

1. <u>Preserve Program Order</u>: 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 returns the value written by P

2. <u>Coherent view of memory</u>: 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

3. <u>Write serialization</u>: 2 writes to same location by any 2 processors are seen in the same order by all processors

   - For example, if the values 1 and then 2 are written to a location, processors can never read the value of the location as 2 and then later read it as 1

   - Otherwise, a processor could keep value 1 indefinitely

11

Enforcing Coherence

- Coherent caches provide:
    - *Migration*: movement of data to caches
    - *Replication*: multiple copies of data in caches

- Cache coherence protocols
    - Snooping each core snoop the bus (L1   L2   bus)
        - Each core tracks sharing status of each block
    - Directory based
        - Sharing status of each block kept in one location

Snoopy Coherence Protocols

- **Write invalidate** with write back cache
  - On write, invalidate all other copies to ensure exclusive access
  - Use bus itself to serialize
    - Write cannot complete until bus access is obtained

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-back cache   Processor A   block   Processor B   ,
memory (shared)   write   .

- **Write update**
  - On write, update all copies

13

Snoopy Cache-Coherence Protocols
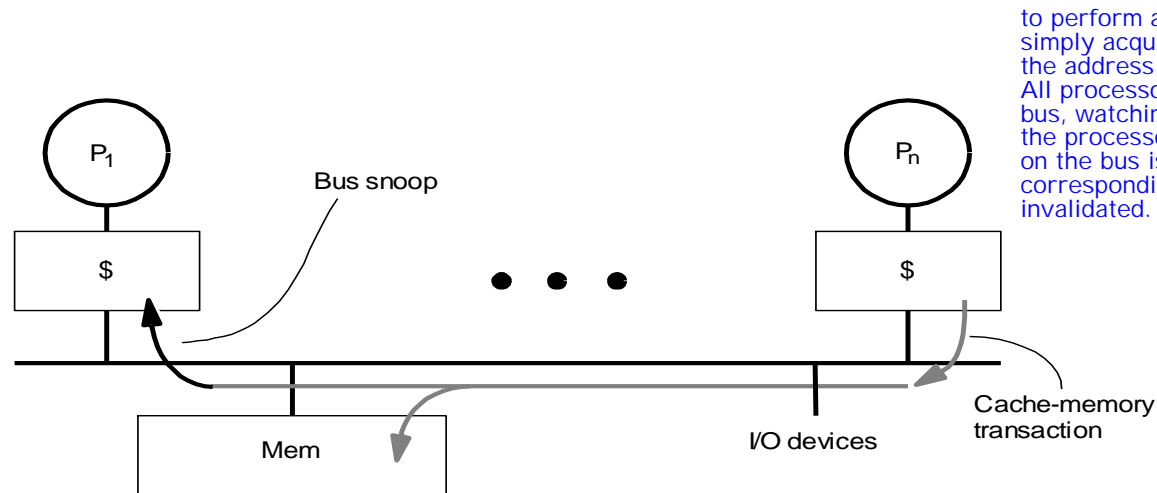
to perform an invalidate, the processor simply acquires bus access and broadcasts the address to be invalidated on the bus. All processor continuously snoop on the bus, watching the addresses. the processors check whether the address on the bus is in their cache. If so, the corresponding data in the cache are invalidated.

- Cache Controller "snoops" all transactions on the shared medium (bus or switch)
    - relevant transaction if it contains requested block
    - take action to ensure coherence
        - invalidate, update, or supply value
    - depends on state of the block and the protocol

14

Snoopy Coherence Protocols

- **Locating an item when a read miss occurs**
  - In write-back cache, the updated value must be sent to the requesting processor

    write-through cache – it is easy to find the recent value of a data item. since all written data are always sent to the memory, from which the most recent value of a data item can always be fetched.
    write-back cache – the most recent value of a data item can be in a private cache. each processor snoops every address placed on the shared bus. If a processor finds that it has a dirty copy of the requested cache block, it provides that cache block in response to the read request and causes the memory access to be aborted

- **Cache lines marked as shared or exclusive/modified**
  - Only writes to shared lines need an invalidate broadcast
    - After this, the line is marked as exclusive

Snoopy Coherence Protocols

Request	Source	State of addressed cache block	Type of cache action	Function and explanation
Read hit	Processor	Shared or modified	Normal hit	Read data in local cache.
Read miss	Processor	Invalid	Normal miss	Place read miss on bus.
Read miss	Processor	Shared	Replacement	Address conflict miss: place read miss on bus.
Read miss	Processor	Modified	Replacement	Address conflict miss: write-back block, then place read miss on bus.
Write hit	Processor	Modified	Normal hit	Write data in local cache.
Write hit	Processor	Shared	Coherence	Place invalidate on bus. These operations are often called upgrade or *ownership* misses, since they do not fetch the data but only change the state.
Write miss	Processor	Invalid	Normal miss	Place write miss on bus.
Write miss	Processor	Shared	Replacement	Address conflict miss: place write miss on bus.
Write miss	Processor	Modified	Replacement	Address conflict miss: write-back block, then place write miss on bus.
Read miss	Bus	Shared	No action	Allow shared cache or memory to service read miss.
Read miss	Bus	Modified	Coherence	Attempt to share data: place cache block on bus and change state to shared.
Invalidate	Bus	Shared	Coherence	Attempt to write shared block; invalidate the block.
Write miss	Bus	Shared	Coherence	Attempt to write shared block; invalidate the cache block.
Write miss	Bus	Modified	Coherence	Attempt to write block that is exclusive elsewhere; write-back the cache block and make its state invalid in the local cache.

16

Snoopy Coherence Protocols

The state in each node represents the state of the selected private cache block specified by the processor or bus request

core cache block data

the block in the private cache is potentially shared

the block has been updated in the private cache.

read miss   block   local cache   clean
P1   exclusive   block   P2   read miss
P1   dirty block   write back   P2   block
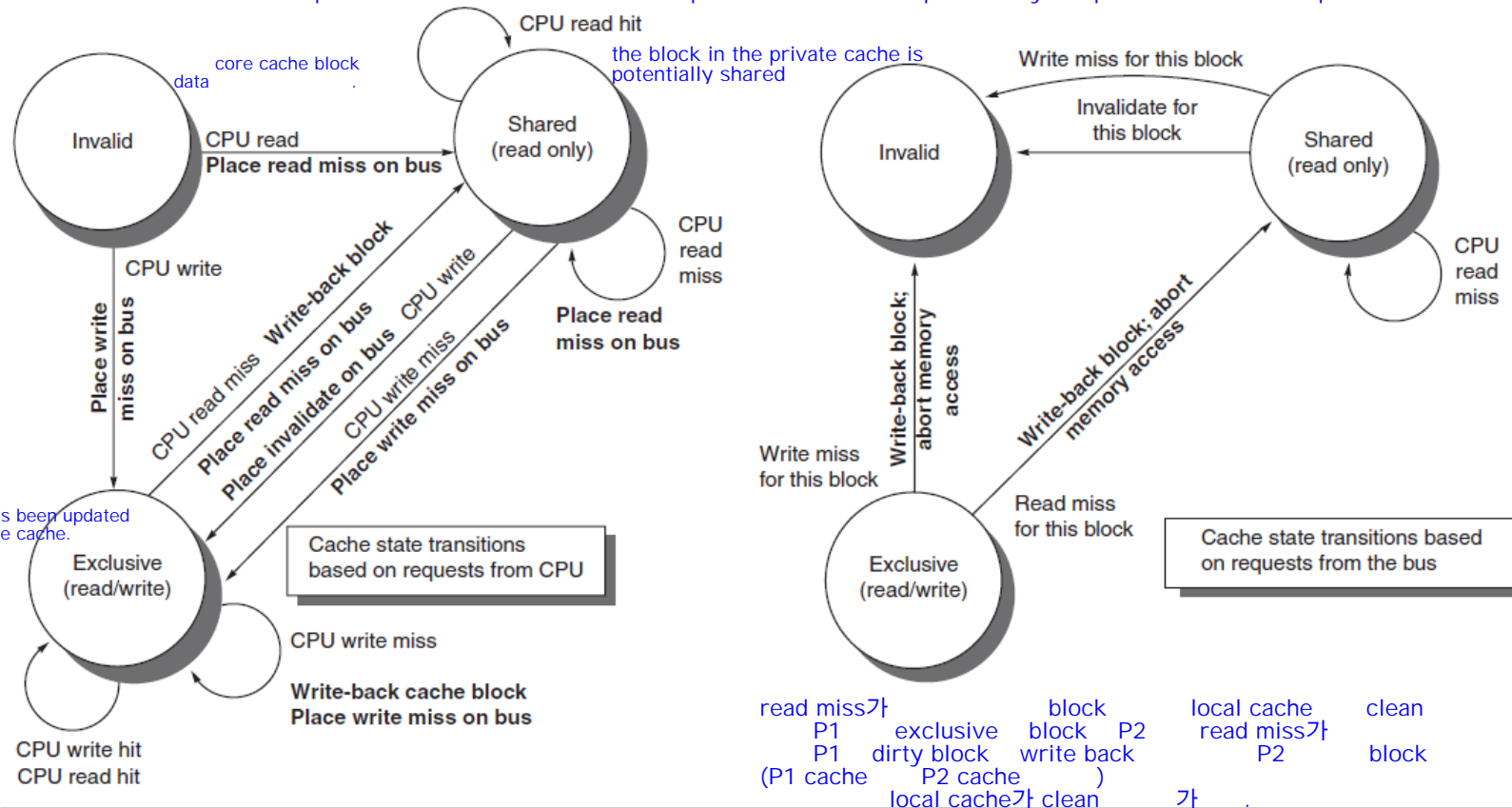(P1 cache   P2 cache   )
local cache   clean

Figure 5.6 A write invalidate, cache coherence protocol for a private write-back cache showing the states and state transitions for each block in the cache. The cache states are shown in circles, with any access permitted by the block in the shared state generates an invalidate. Whenever a bus transaction occurs, all private caches that contain the cache block specified in the bus transaction take the action dictated by the right half of the diagram. The protocol assumes that memory (or a shared cache) provides data on a read miss for a block that is clean in all local caches. In actual implementations, these two sets of state diagrams are combined. In practice, there are many subtle variations on invalidate protocols, including the introduction of the exclusive unmodified state, as to whether a processor or memory provides data on a miss. In a multicore chip, the shared cache (usually L3, but sometimes L2) acts as the equivalent of memory, and the bus is the bus between the private caches of each core and the shared cache, which in turn interfaces to the memory.

Snoopy Coherence Protocols

- Complications for the basic MSI protocol: (modified, shared, invalid)
  - Operations are not atomic
    - E.g. detect miss, acquire bus, receive a response
    - Creates possibility of deadlock and races
    - One solution: processor that sends invalidate can hold bus until other processors receive the invalidate

- Extensions:
  - Add exclusive state to indicate clean block in only one cache (MESI protocol) (modified, exclusive, shared, invalied)
    - Prevents needing to write invalidate on a write
  - Owned state (MOESI) (modified, owned, exclusive, shared, invalied)
    - Associated block is owned by that cache and out-of-date in memory, which is responsible for replacement and providing data on a miss

In MSI and MESI protocols, when there is an attempt to share a block in the Modified state, the state is changed to Shared (in both the original and newly sharing cache), and the block must be written back to memory.
In MOESI protocol, the block can be changed from the Modified to Owned state in the original cache without writing it to memory.

Coherence Protocols: Extensions

- **Shared memory bus and snooping bandwidth is bottleneck for scaling symmetric multiprocessors**
    - Duplicating tags
    - Place directory in outermost cache
    - Use crossbars or point-to-point networks with banked memory