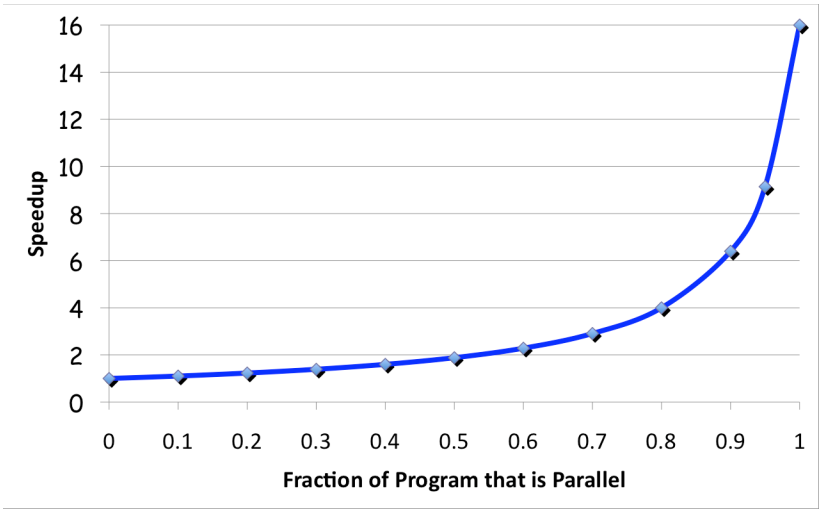
Lecture 24: Parallelism

- Administration
  - Take QUIZ 18 over P&H 7.6-13, before 11:59pm today
  - Project: Cache Simulator, Due April 29, 2010
- Last Time
  - Where do architectures exploit parallelism?
  - What are the implications for programming models?
  - What are the implications for communication?
  - … for caching?
- Today
  - What are the implications for caching and communication?

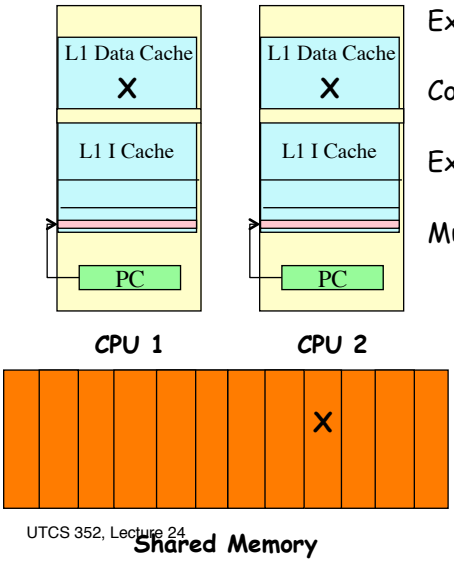
Remember Amdahl's Law



Granularity of Parallelism Too Hot, Too Cold, or Just Right?

- Fine grain parallel architectures
  - Lots of instruction & data communication
    - Instructions/data all on the same chip
    - Communication latency is very low, order of cycles
- Coarse grain parallel architectures
  - Lots and lots of independent data & work
  - Rarely or never communicate, because communication is very, very expensive
- Multicore is betting there are applications with
  - Medium grain parallelism (i.e., threads, processes, tasks)
  - Not too much data, so it wont swamp the memory system
  - Not too much communication (100 to 1000s of cycles across chip through the memory hierarchy)

Shared Memory Multicore Programming Model



Example: Two threads running on two CPUs

Communicate implicitly by accessing shared global state

Example: both threads access shared variable X

Must synchronize accesses to X

```
thread 1                    thread 2
withdrawal(wdr) {           deposit(dep) {
    lock(l)                     lock(l)
    if X > wdr                  X = X + dep
      X = X – wdr               bal = X
    bal = X                     unlock(l)
    unlock(l)
    return bal
```

Shared Memory Cache Coherence Problem

- Two threads share variable X
- Hardware
  - Two CPUs, write-through caches

Time step	Event	CPU 1's cache	CPU 2's cache	Memory
0				10
1	CPU 1 reads X	10		10
2	CPU 2 reads X	10	10	10
3	CPU 1 writes 1 to X	5	10	5

Memory Consistency Models

- When are writes seen by other processors?
  - "Seen" means a read returns the written value
  - Can not be instantaneous!
- Assumptions
  - A write completes only when all processors have seen it
  - A processor does not reorder writes with other accesses
- Consequence
  - P writes X then writes Y
    ⇒ all processors that see new Y also see new X
  - Processors can reorder reads, but not writes

Cache Coherence Defined

Sequential Coherence

Reads return most recently written value

Formally

- P writes X; P reads X (no intervening writes)
  ⇒ read returns written value
- P_1 writes X; P_2 reads X
  ⇒ read returns written value
  - c.f. CPU 2 reads X = 5 after step 3 in example
- P_1 writes X, P_2 writes X
  ⇒ all processors see writes in the same order
  - End up with the same final value for X

Cache Coherence Protocols

Operations performed by multiprocessors to ensure cache coherence

- Migration of data to local caches
  - Reduces bandwidth for shared memory
- Find the correct data

Bus Based Snooping protocols

- Popular, but scales poorly

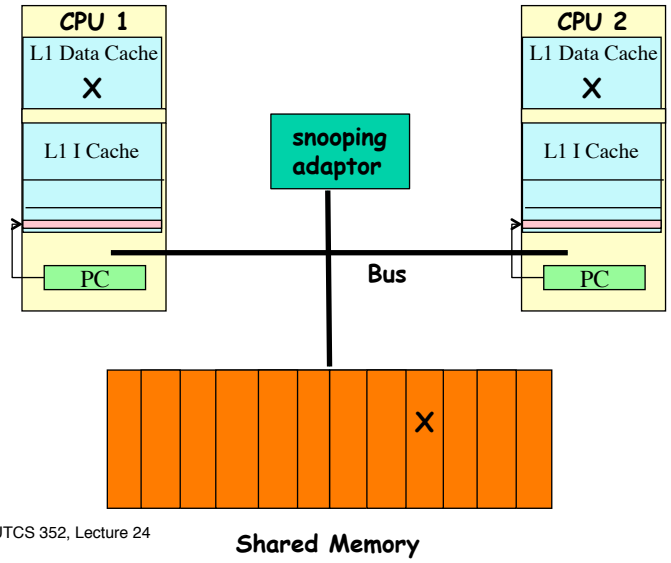
Hierarchical & Directory-based protocols

- Scale better, but require more state and complexity

Bus Based Snooping Protocols

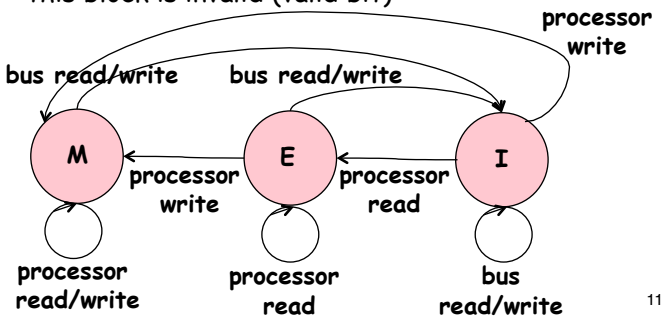
- Caches connected to a bus
- Caches "snoop" bus to know what the other caches are doing
- Act on other cache actions
- Cache line bits for "state" of sharing
- States: **Modified, Exclusive, Shared, Invalid**
- Finite state machine logic determines actions based on current state and request (read or write)

Bus Based Protocol Architecture



MEI: Most Basic Snooping Protocol

- Lots of messages, invalidations, does not support write back
- Protocol letters MEI encode cache block (line) state
  - **Modified** – this block is modified in cache (dirty bit)
  - **Exclusive** – this block is a read only copy
  - **Invalid** – this block is invalid (valid bit)



MEI Example

Only one cache ever holds a block in M or E state
Supports write back at invalidation time

CPU activity	Bus activity	CPU 1's cache	CPU 2's cache	Memory
				10
CPU 1 reads X	Cache miss for X			10
CPU 2 reads X	Cache miss for X			10
CPU 1 writes 5 to X	Invalidate for X			
CPU 2 read X	Cache miss for X			

MEI Example

Only one cache ever holds the data in either M or E mode

Supports write back at invalidation time

CPU activity	Bus activity	CPU 1's cache	CPU 2's cache	Memory
				10
CPU 1 reads X	Cache miss for X	10 (E)		10
CPU 2 reads X	Cache miss for X	10 (I)	10 (E)	10
CPU 1 writes 5 to X	Invalidate for X	5 (M)	10 (I)	10
CPU 2 read X	Cache miss for X	5 (I)	5 (X)	5

MSI: Better, but still basic snooping protocol

- Used in SGI 4D processor
- Supports sharing & write back
- Supports readX: read exclusive, only one cache can contain the line
- Protocol letters MSI encode cache block (line) state

Modified – block modified in cache & inconsistent with backing store (memory or cache). Cache must write block back on eviction.

Shared – block in at least one cache & unmodified

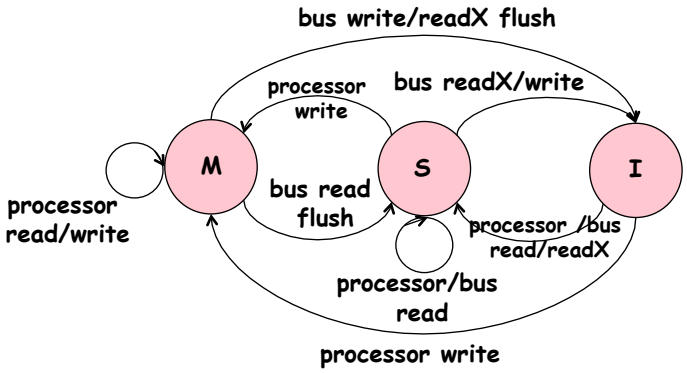
Invalid – block invalid

MSI State Diagram

Modified – block modified in cache & inconsistent with backing store (memory or cache). Cache must write block back on eviction.

Shared – block in at least one cache & unmodified

Invalid – block invalid



MSI Example

Multiple caches may hold a block in S state
Only one cache holds a block in M state
Supports write back at invalidation time

CPU activity	Bus activity	CPU 1's cache	CPU 2's cache	Memory
				10
CPU 1 reads X	Cache miss for X			10
CPU 2 reads X	Cache miss for X			10
CPU 1 writes 5 to X	Invalidate for X			
CPU 2 read X	Cache miss for X			

MSI Example

Multiple caches may hold a block in S state
Only one cache holds a block in M state
Supports write back at invalidation time

CPU activity	Bus activity	CPU 1's cache	CPU 2's cache	Memory
				10
CPU 1 reads X	Cache miss for X	10 (S)		10
CPU 2 reads X	Cache miss for X	10 (S)	10 (S)	10
CPU 1 writes 5 to X	Invalidate for X	5 (M)	10 (I)	10
CPU 2 read X	Cache miss for X	5 (I)	5 (S)	5

MESI & MEOSI Snooping Protocol

Modified – block modified in cache & inconsistent with backing store (memory or cache). Cache must write block back on eviction.

Owner – cache owns block & can modify it without a bus message

Exclusive – block only in this cache & unmodified

Shared – block in at least one other cache & unmodified

Invalid – block invalid

Industry Cache Coherence Implementations

Intel: MESI

AMD: MOESI

Sun Microsystems: JBus (MOESI)

Granularity of Parallelism = Cost of Communication Too Hot, Too Cold, or Just Right?

- Fine grain parallel architectures
  - Instructions/data all on the same chip
  - Communication latency is very low, order of 1-2 cycles
- Coarse grain parallel architectures
  - Lots and lots of independent data & work
  - Rare communication, communication is very expensive
- Multicore
  - Medium grain parallelism
  - Not too much data & data with good locality, so threads wont swamp the memory system
  - Not too much communication (100 to 1000s of cycles across chip through the memory hierarchy)
  - Shared data in caches leads to lots of communication
  - Explicitly send messages instead of shared memory? Is that programming model too hot?

Summary

- Parallelism
  - Granularities
  - Implications for programming models
  - Implications for communication
  - Implications for caches
- Next Time
  - Doug Burger, EDGE architectures
    - Parallelism and single thread performance
  - Optional readings
    - ``An Evaluation of the TRIPS Computer System,'' M. Gebhart et al., The ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 1-12, March, 2009. Best paper award.
    - ``Scaling to the End of Silicon with EDGE Architectures,'' Burger et al., IEEE Computer, pp. 44 - 55, July, 2004.