Review: Centralized Shared-Memory Architecture

- Small processor counts makes it possible
  - that processors share one a single centralized memory
  - to interconnect the processors and memory by a bus

Review: Distributed Memory Machines

- Nodes include processor(s), some memory, typically some IO, and interface to an interconnection network

Small-Scale—Shared Memory

- Caches serve to:
  - Increase bandwidth versus bus/memory
  - Reduce latency of access
  - Valuable for both private data and shared data
- What about cache consistency?

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>$A$</th>
<th>$B$</th>
<th>$X$ (memory)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>CPU A: R</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>CPU B: R</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>CPU A: W</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
What Does Coherency Mean?

Informally:
- “Any read of a data item must return the most recently written value”
- this definition includes both coherence and consistency
  - coherence: what values can be returned by a read
  - consistency: when a written value will be returned by a read

Memory system is coherent if
- a read(X) by P1 that follows a write(X) by P1, with no writes of X by another processor occurring between these two events, always returns the value written by P1
- a read(X) by P1 that follows a write(X) by another processor, returns the written value if the read and write are sufficiently separated and no other writes occur between
- writes to the same location are serialized: two writes to the same location by any two CPUs are seen in the same order by all CPUs

Potential HW Coherence Solutions

Snooping Solution (Snoopy Bus):
- every cache that has a copy of the data also has a copy of the sharing status of the block
- Processors snoop to see if they have a copy and respond accordingly
- Requires broadcast, since caching information is at processors
- Works well with bus (natural broadcast medium)
- Dominates for small scale machines (most of the market)

Directory-Based Schemes (discuss later)
- Keep track of what is being shared in 1 centralized place (logically)
- Distributed memory => distributed directory for scalability (avoids bottlenecks)
- Send point-to-point requests to processors via network
- Scales better than Snooping
- Actually existed BEFORE Snooping-based schemes

Basic Snoopy Protocols

Write Invalidate Protocol
- A CPU has exclusive access to a data item before it writes that item
- Write to shared data: an invalidate is sent to all caches which snoop and invalidate any copies
- Read Miss:
  - Write-through: memory is always up-to-date
  - Write-back: snoop in caches to find most recent copy

Write Update Protocol (typically write through):
- Write to shared data: broadcast on bus, processors snoop, and update any copies
- Read miss: memory is always up-to-date

Write serialization: bus serializes requests!
- Bus is single point of arbitration

Write Invalidate versus Update

Multiple writes to the same word with no intervening reads
- Update: multiple broadcasts

For multiword cache blocks
- Update: each word written in a cache block requires a write broadcast
- Invalidate: only the first write to any word in the block requires an invalidation

Update has lower latency between write and read
Snooping Cache Variations

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclusive</td>
<td>Owned Exclusive</td>
<td>Private Dirty</td>
<td>Modified (private,≠Memory)</td>
</tr>
<tr>
<td>Shared</td>
<td>Owned Shared</td>
<td>Private Clean</td>
<td>Exclusive (private,≠Memory)</td>
</tr>
<tr>
<td>Invalid</td>
<td>Shared</td>
<td>Shared</td>
<td>Shared (shared,≠Memory)</td>
</tr>
<tr>
<td>Invalid</td>
<td>Invalid</td>
<td>Invalid</td>
<td>Invalid</td>
</tr>
</tbody>
</table>

Owner can update via bus invalidate operation
Owner must write back when replaced in cache
If read sourced from memory, then Private Clean
If read sourced from other cache, then Shared
Can write in cache if held private clean or dirty

An Example Snoopy Protocol

- Invalidation protocol, write-back cache
- Each block of memory is in one state:
  - Clean in all caches and up-to-date in memory (Shared)
  - OR Dirty in exactly one cache (Exclusive)
  - OR Not in any caches
- Each cache block is in one state (track these):
  - Shared : block can be read
  - OR Exclusive : cache has only copy, its writeable, and dirty
  - OR Invalid : block contains no data
- Read misses: cause all caches to snoop bus
- Writes to clean line are treated as misses

Snoopy-Cache State Machine-I

State machine for CPU requests for each cache block

CPU Read hit → CPU Read → Shared (read-only)
CPU Write Miss on bus → Exclusive (read/write)
CPU Read miss on bus → Place Read miss on bus
CPU Read hit → Exclusive (read/write)
CPU Read miss on bus → Place Read miss on bus
CPU Write Miss on bus → CPU Write Miss on bus
CPU Write hit → Exclusive (read/write)
CPU Write hit → Write back block, Place Read miss on bus
CPU Read hit → Exclusive (read/write)
CPU Read miss on bus → Place Read miss on bus
CPU Write Miss on bus → CPU Write Miss on bus
CPU Write hit → Exclusive (read/write)
CPU Write hit → Write back cache block, Place Read miss on bus

Snoopy-Cache State Machine-II

State machine for bus requests for each cache block

Invalid → Write miss for this block
Invalid → Read miss for this block
Invalid → Exclusive (read/write)
Write miss for this block
Write Back Block; (abort memory access)
Read miss for this block
Write Back Block; (abort memory access)
Snoopy-Cache State Machine-III

State machine for CPU requests for each cache block and for bus requests for each cache block.

- Invalid: Write miss for this block.
- Shared (read/only): Place read miss on bus.
- Exclusive (read/write): Write miss for this block.
- Write Back: Block; (abort memory access).
- Write back block: Place read miss on bus.
- CPU Read hit: Place Write Miss on Bus.
- CPU Write hit: CPU Write Miss.
- Write Back Block: (abort memory access).
- Write Back cache block: Place write miss on bus.

Example: Step 1

- Processor 1: Write 10 to A1
- Processor 2: Read A1
- Processor 2: Write 20 to A1
- Processor 2: Write 40 to A2

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2.

Example: Step 2

- Processor 1: Write 10 to A1
- Processor 2: Read A1
- Processor 2: Write 20 to A1
- Processor 2: Write 40 to A2

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2.
## Example: Step 3

<table>
<thead>
<tr>
<th>Step</th>
<th>State</th>
<th>Addr</th>
<th>Value</th>
<th>State</th>
<th>Addr</th>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Write 10 to A1</td>
<td>Exc.</td>
<td>A1</td>
<td>10</td>
<td>Excl.</td>
<td>A1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Read A1</td>
<td>Excl.</td>
<td>A1</td>
<td>10</td>
<td>Excl.</td>
<td>A1</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Write 20 to A1</td>
<td>Inv.</td>
<td>A1</td>
<td>20</td>
<td>Excl.</td>
<td>A1</td>
<td>20</td>
</tr>
</tbody>
</table>

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2.

### Implementation Complications

- **Write Races:**
  - Cannot update cache until bus is obtained
  - Otherwise, another processor may get bus first, and then write the same cache block!
  - Two step process:
    - Arbitrate for bus
    - Place miss on bus and complete operation
  - If miss occurs to block while waiting for bus, handle miss (invalidate may be needed) and then restart
  - Split transaction bus:
    - Bus transaction is not atomic: can have multiple outstanding transactions for a block
    - Multiple misses can interleave, allowing two caches to grab block in the Exclusive state
    - Must track and prevent multiple misses for one block
  - Must support interventions and invalidations
Implementing Snooping Caches

- Multiple processors must be on bus, access to both addresses and data
- Add a few new commands to perform coherency, in addition to read and write
- Processors continuously snoop on address bus
- If address matches tag, either invalidate or update
- Since every bus transaction checks cache tags, could interfere with CPU just to check:
  - Solution 1: duplicate set of tags for L1 caches just to allow checks in parallel with CPU
  - Solution 2: L2 cache already duplicate, provided L2 obeys inclusion with L1 cache
- Block size, associativity of L2 affects L1

Bus serializes writes, getting bus ensures no one else can perform memory operation

On a miss in a write back cache, may have the desired copy and its dirty, so must reply

- Add extra state bit to cache to determine shared or not
- Add 4th state (MESI)

MESI: CPU Requests

- CPU Read hit
- CPU Read miss
- CPU Write hit
- CPU Write miss

MESI: Bus Requests

- BusRdEx
- BusInv
- BusRd / => Sh
- Modified (read/write)
Fundamental Issues

- 3 Issues to characterize parallel machines
  - 1) Naming
  - 2) Synchronization
  - 3) Performance: Latency and Bandwidth (covered earlier)

Fundamental Issue #1: Naming

- Naming: how to solve large problem fast
  - what data is shared
  - how it is addressed
  - what operations can access data
  - how processes refer to each other
- Choice of naming affects code produced by a compiler; via load where just remember address or keep track of processor number and local virtual address for msg. passing
- Choice of naming affects replication of data; via load in cache memory hierarchy or via SW replication and consistency

Fundamental Issue #1: Naming

- Global physical address space:
  - any processor can generate, address and access it in a single operation
  - memory can be anywhere:
    - virtual addr. translation handles it
- Global virtual address space: if the address space of each process can be configured to contain all shared data of the parallel program
- Segmented shared address space:
  - locations are named
  - <process number, address> uniformly for all processes of the parallel program

Fundamental Issue #2: Synchronization

- To cooperate, processes must coordinate
- Message passing is implicit coordination with transmission or arrival of data
- Shared address
  - additional operations to explicitly coordinate:
    - e.g., write a flag, awaken a thread, interrupt a processor
Summary: Parallel Framework

- **Layers:**
  - **Programming Model:**
    - Multiprogramming: lots of jobs, no communication
    - Shared address space: communicate via memory
    - Message passing: send and receive messages
    - Data Parallel: several agents operate on several data sets simultaneously and then exchange information globally and simultaneously (shared or message passing)
  - **Communication Abstraction:**
    - Shared address space: e.g., load, store, atomic swap
    - Message passing: e.g., send, receive library calls
    - Debate over this topic (ease of programming, scaling)
      => many hardware designs 1:1 programming model

Distributed Directory MPs

- **Directory Protocol:**
  - Similar to Snoopy Protocol: Three states
    - Shared: ≥ 1 processors have data, memory up-to-date
    - Uncached (no processor has it; not valid in any cache)
    - Exclusive: 1 processor (owner) has data; memory out-of-date
  - In addition to cache state, must track which processors have data when in the shared state (usually bit vector, 1 if processor has copy)
  - Keep it simple(r):
    - Writes to non-exclusive data
    => write miss
    - Processor blocks until access completes
    - Assume messages received and acted upon in order sent

- **Terms:** typically 3 processors involved
  - Local node where a request originates
  - Home node where the memory location of an address resides
  - Remote node has a copy of a cache block, whether exclusive or shared

- **Example messages on next slide:**
P = processor number, A = address
### Directory Protocol Messages

<table>
<thead>
<tr>
<th>Message type</th>
<th>Source</th>
<th>Destination</th>
<th>Msg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read miss</td>
<td>Local cache</td>
<td>Home directory</td>
<td>P, A</td>
</tr>
<tr>
<td>Write miss</td>
<td>Local cache</td>
<td>Home directory</td>
<td>P, A</td>
</tr>
<tr>
<td>Invalidate</td>
<td>Home directory</td>
<td>Remote caches</td>
<td>A</td>
</tr>
<tr>
<td>Fetch</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
</tr>
<tr>
<td>Fetch/Invalidate</td>
<td>Home directory</td>
<td>Remote cache</td>
<td>A</td>
</tr>
<tr>
<td>Data value reply</td>
<td>Home directory</td>
<td>Local cache</td>
<td>Data</td>
</tr>
<tr>
<td>Data write-back</td>
<td>Remote cache</td>
<td>Home directory</td>
<td>A, Data</td>
</tr>
</tbody>
</table>

**Example Messages:***
- **Read miss**: Processor P reads data at address A; make P a read sharer and arrange to send data back to the home directory.
- **Write miss**: Processor P writes data at address A; make P the exclusive owner and arrange to send data back to the home directory.
- **Invalidate**: Invalidate a shared copy at address A.
- **Fetch**: Fetch the block at address A and send it to its home directory.
- **Fetch/Invalidate**: Fetch the block at address A and send it to its home directory; invalidate the block in the cache.
- **Data value reply**: Return a data value from the home memory (read miss response).
- **Data write-back**: Write-back a data value for address A (invalidate response).

### State Transition Diagram for an Individual Cache Block in a Directory Based System

- States identical to snoopy case; transactions very similar.
- Transitions caused by read misses, write misses, invalidates, data fetch requests.
- Generates read miss & write miss msg to home directory.
- Write misses that were broadcast on the bus for snooping => explicit invalidate & data fetch requests.
- Note: on a write, a cache block is bigger, so need to read the full cache block.

### State Transition Diagram for the Directory

- Same states & structure as the transition diagram for an individual cache.
- 2 actions: update of directory state & send msgs to satisfy requests.
- Tracks all copies of memory block.
- Also indicates an action that updates the sharing set, Sharers, as well as sending a message.
Directory State Machine

- State machine for Directory requests for each memory block
- Uncached state if in memory
  - Data Write Back: Sharers = {}
    - (Write back block)
  - Miss: Sharers = (P); Fetch/Invalidate Data Value Reply msg to remote cache
- Exclusive (read/write)
  - Read miss: Sharers += {P}; send Data Value Reply msg to remote cache
  - Write Miss: Sharers = {P}; send Fetch/Invalidate; send Data Value Reply msg to remote cache
- Shared (read only)
  - Read miss: Sharers += {P}; send Data Value Reply msg
  - Write Miss: send Invalidate to Sharers; then Sharers = {P}; send Data Value Reply msg

- Read miss:
  - Sharers += {P}; send Data Value Reply msg

Uncached state

Data Write Back:
Sharers = {}