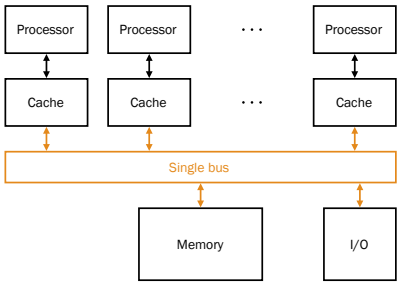
CSE 240B
Advanced Computer Architecture

Dean Tullsen

Multiprocessors and Multiprocessing

Multiprocessors

- why would you want a multiprocessor?
- what things can it do well?
- What things can't it do well?
- What things can it do that a *bunch of computers* can't do?
- How much are you willing to pay?



Classifying Multiprocessors

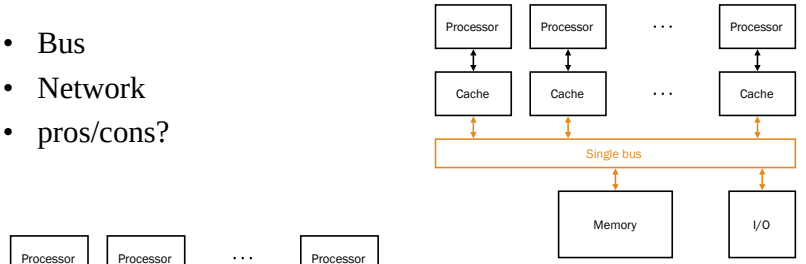
- Flynn Taxonomy
- Interconnection Network
- Memory Topology
- Programming Model

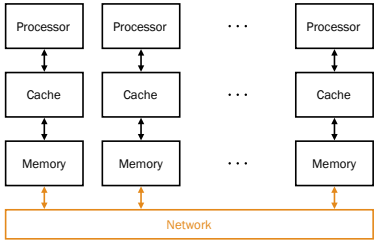
Flynn Taxonomy

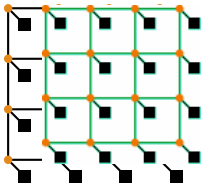
- SISD (Single Instruction Single Data)
  - Uniprocessors
- MISD (Multiple Instruction Single Data)
  - ???
- SIMD (Single Instruction Multiple Data)
  - Examples: Illiac-IV, CM-2
    - » Simple programming model
    - » Low overhead
    - » All custom
- MIMD (Multiple Instruction Multiple Data)
  - Examples: many, nearly all modern MPs
    - » Flexible
    - » *Use off-the-shelf micros*

Interconnection Network

- Bus
- Network
- pros/cons?

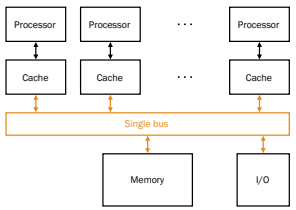


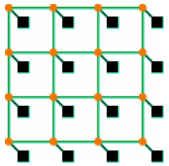


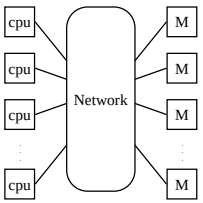


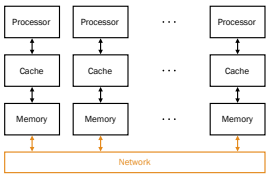
Memory Topology

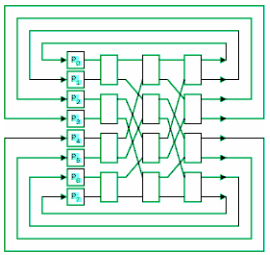
- UMA (Uniform Memory Access)
- NUMA (Non-uniform Memory Access)
- pros/cons?





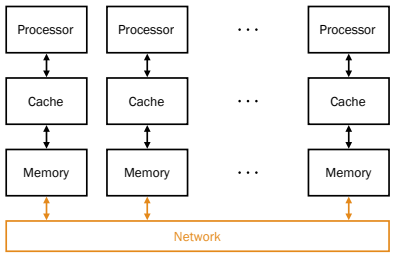






Programming Model

- Shared Memory -- every processor can name every address location
- Message Passing -- each processor can name only it's local memory. Communication is through explicit messages.



shared memory architecture with network interconnection sometimes called *Distributed Shared Memory (DSM)*

Parallel Programming -- Review

i = 47

Processor A

index = i++;

Processor B

index = i++;

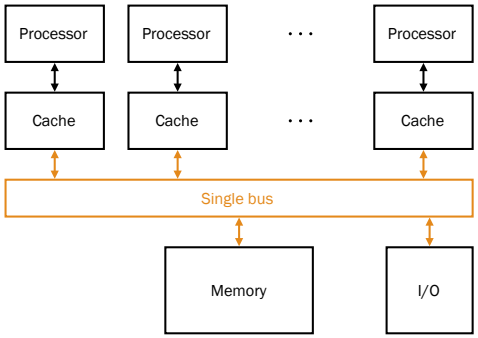
- Shared-memory programming requires synchronization to provide mutual exclusion and prevent race conditions
  - locks (semaphores)
  - barriers

Communication Models

- Shared Memory
  - Processors communicate with shared address space
  - Easy on small-scale machines
  - Advantages:
    - Model of choice for uniprocessors, small-scale MPs
    - Ease of programming
    - Lower latency
    - Easier to use hardware controlled caching
- Message passing
  - Processors have private memories, communicate via messages
  - Advantages:
    - Less hardware, easier to design
    - Focuses attention on costly *non-local* operations
- Can support either model on either HW base

Small-Scale Multiprocessors — Shared Memory

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



What Does Coherence Mean?

- Informally:
  - Any read must return the most recent write
  - Too strict and very difficult to implement
- Better:
  - A processor sees its own writes to a location in the correct order.
  - Any write must eventually be seen by a read
  - All writes are seen in order ("serialization"). Writes to the same location are seen in the same order by all processors.
- Without these guarantees, synchronization doesn't work.

Potential Solutions

- Snooping Solution (Snoopy Bus):
  - Send all requests for unknown data to all processors
  - 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
  - Keep track of what is being shared in one centralized place
  - Distributed memory => distributed directory (avoids bottlenecks)
  - Send point-to-point requests to processors
  - Scales better than Snoop
  - Actually existed BEFORE Snoop-based schemes

Basic Snoopy Protocols

- *Write Invalidate* Protocol:
  - 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:
  - Write to shared data: broadcast on bus, processors snoop, and *update* copies
  - Read miss: memory is always up-to-date
- Write serialization: bus serializes requests
  - Bus is single point of arbitration

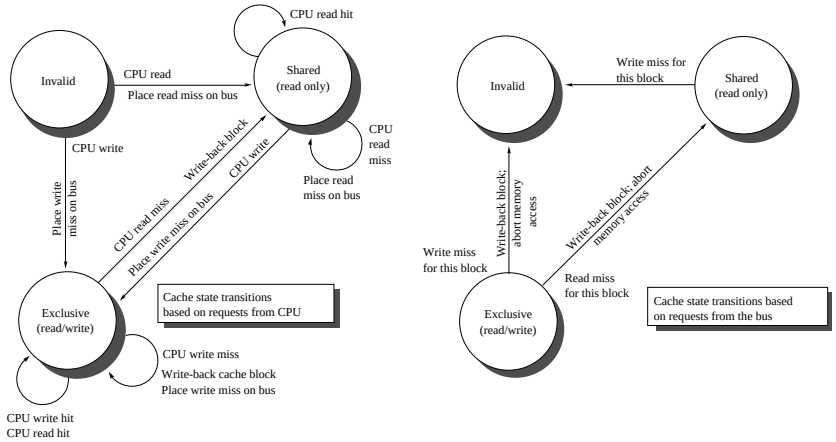
Basic Snoopy Protocols

- Write Invalidate versus Broadcast:
  - Invalidate requires one transaction per write-run
  - Invalidate exploits spatial locality: one transaction per block
  - Broadcast has lower latency between write and read
  - Broadcast: BW (increased) vs. latency (decreased) tradeoff

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
  - Dirty in exactly one cache
  - Not in any caches
- Each cache block is in one state:
  - **(S)**hared: block can be read
  - **(E)**xclusive: cache has only copy, its writeable, and dirty
  - **(I)**nvalid: block contains no data
- Read misses: cause all caches to snoop
- Writes to clean line are treated as misses

Snoopy-Cache State Machine



Example

ESI Protocol

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	E	A1	10	I			WM	1	A1		A1	old
P1: Read A1	E	A1	10									
P2: Read A1							abort RM					
							WB	1	A1	10	A1	10
	S	A1	10	S	A1	10	RM	2	A1	10		
P2: Write 20 to A1	I			E	A1	20	WM	2	A1		A1	10
P2: Write 40 to A2	I			E	A2	40	WB	2	A1	20	A1	20
							WM	2	A2	40	A2	old

Assumes A1 and A2 map to same cache block

Snoopy Cache: State Machine

Extensions:

- Fourth State: Ownership
- Clean exclusive state (no miss for private data on write)
  Illinois Protocol (also MESI)
- Cache-cache transfers

Multiprocessors -- Key Points

- Network vs. Bus
- Message-passing vs. Shared Memory
- Shared Memory is more intuitive, but creates problems for both the programmer (memory consistency, requiring synchronization) and the architect (cache coherence).