CMSC 611: Advanced Computer Architecture

Parallel Computation

Most slides adapted from David Patterson. Some from Mohomed Younis

Data Parallel Languages

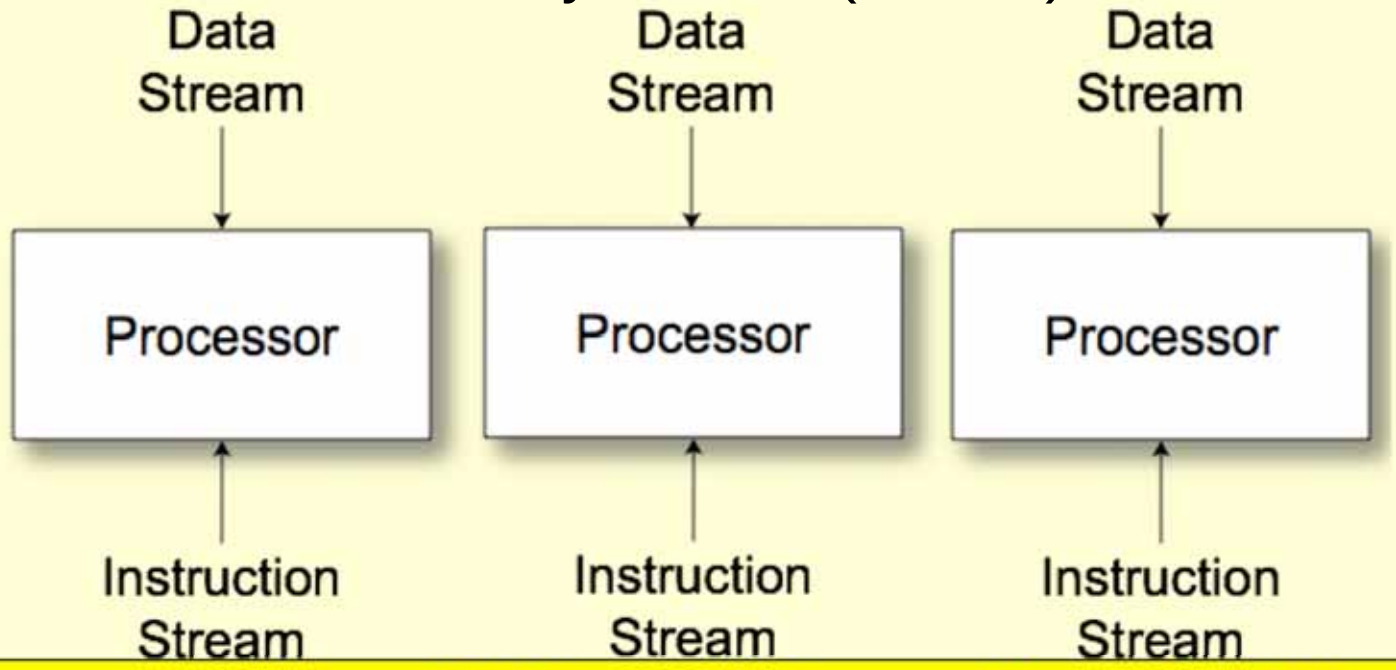
- SIMD programming
  - PE point of view
  - Data: shared or per-PE
    - What data is distributed?
    - What is shared over PE subset
    - What data is broadcast with instruction stream?
  - Data layout: shape [256][256]d;
  - Communication primitives
  - Higher-level operations
    - Prefix sum: $[i]r = \sum_{j \le i} [j]d$
      - 1,1,2,3,4 → 1,1+1=2,2+2=4,4+3=7,7+4=11

Single Program Multiple Data

- Many problems do not map well to SIMD
  - Better utilization from MIMD or ILP
- Data parallel model ⇒ Single Program Multiple Data (SPMD) model
  - All processors execute identical program
  - Same program for SIMD, SISD or MIMD
  - Compiler handles mapping to architecture

MIMD

- Message Passing
- Shared memory/distributed memory
  - Uniform Memory Access (UMA)
  - Non-Uniform Memory Access (NUMA)



Can support either SW model on either HW basis

Message passing

- Processors have private memories, communicate via messages
- Advantages:
  - Less hardware, easier to design
  - Focuses attention on costly non-local operations

Message Passing Model

- Each PE has local processor, data, (I/O)
  - Explicit I/O to communicate with other PEs
  - Essentially NUMA but integrated at I/O vs. memory system
- Free run between Send & Receive
  - Send + Receive = Synchronization between processes (event model)
    - Send: local buffer, remote receiving process/port
    - Receive: remote sending process/port, local buffer

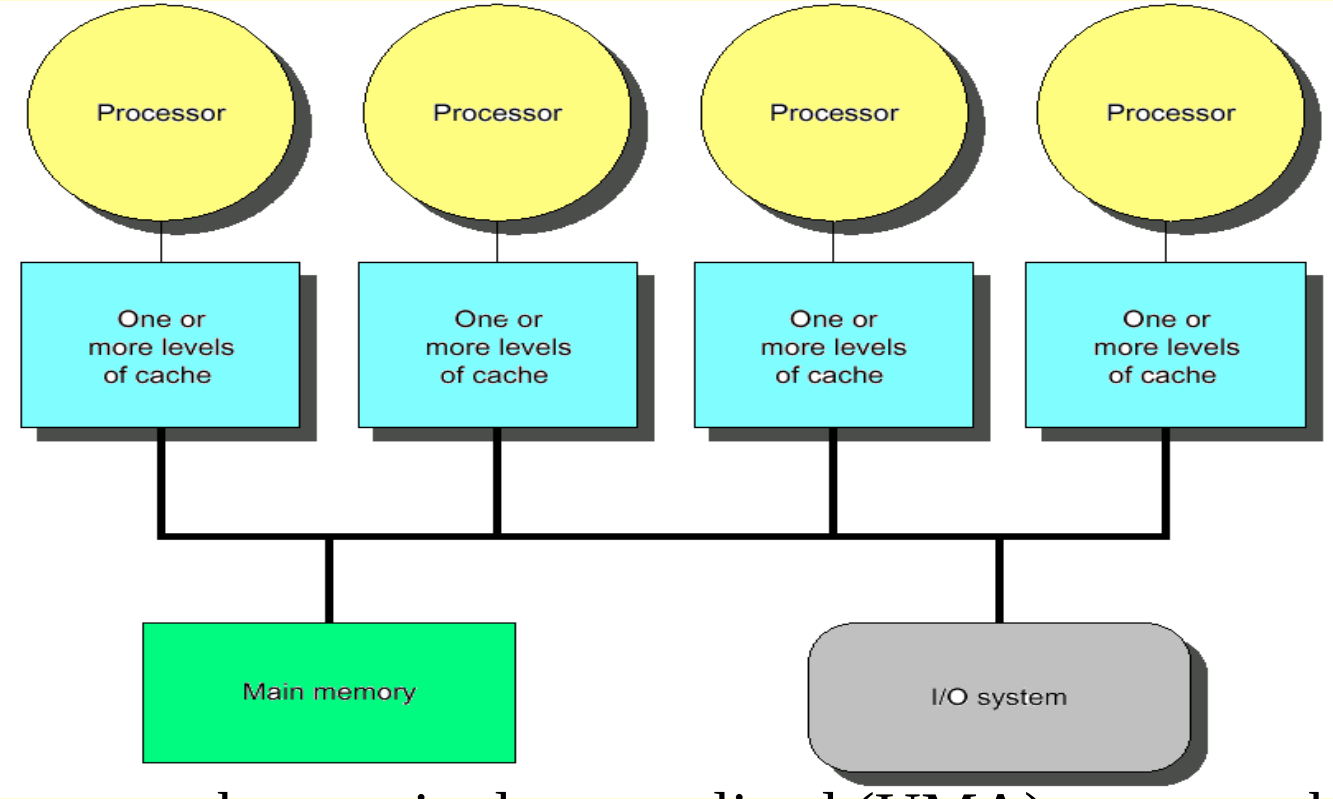
History of message passing

- Early machines
  - Local communication
  - Blocking send & receive
- Later: DMA with non-blocking sends
  - DMA for receive into buffer until processor does receive, and then data is transferred to local memory
- Later still: SW libraries to allow arbitrary communication

Shared Memory

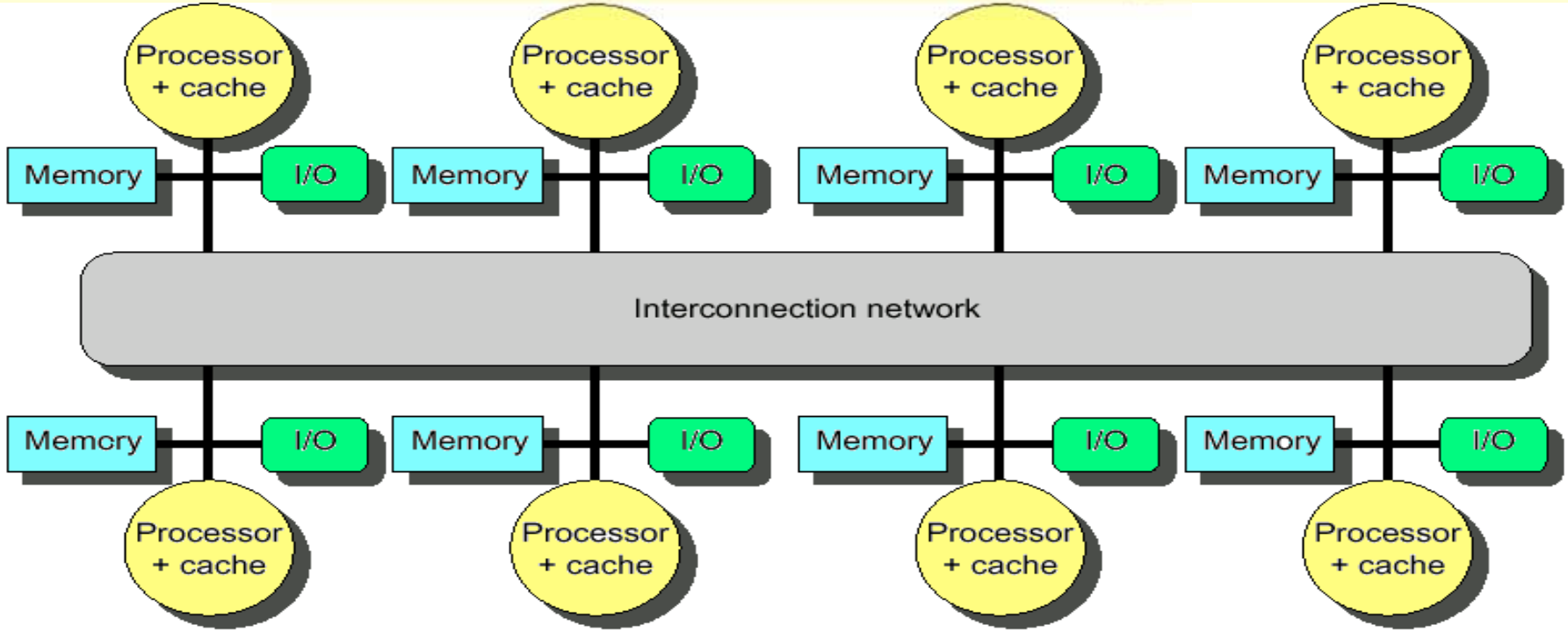
- Processors communicate with shared address space
- Easy on small-scale machines
- Advantages:
  - Model of choice for uniprocessors, small-scale multiprocessor
  - Ease of programming
  - Lower latency
  - Easier to use hardware controlled caching
  - Difficult to handle node failure

Centralized Shared Memory



- Processors share a single centralized (UMA) memory through a bus interconnect
- Feasible for small processor count to limit memory contention
- Centralized shared memory architectures are the most common form of MIMD design

Distributed Memory



- Uses physically distributed (NUMA) memory to support large processor counts (to avoid memory contention)
- Advantages
  - Allows cost-effective way to scale the memory bandwidth
  - Reduces memory latency
- Disadvantage
  - Increased complexity of communicating data

Shared Address Model

- Physical locations
  - Each PE can name every physical location in the machine
- Shared data
  - Each process can name all data it shares with other processes

Shared Address Model

- Data transfer
  - Use load and store, VM maps to local or remote location
  - Extra memory level: cache remote data
  - Significant research on making the translation transparent and scalable for many nodes
    - Handling data consistency and protection challenging
    - Latency depends on the underlying hardware architecture (bus bandwidth, memory access time and support for address translation)
    - Scalability is limited given that the communication model is so tightly coupled with process address space

Three Fundamental Issues

- 1: 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
  - Just remember and load address or keep track of processor number and local virtual address for message passing
- Choice of naming affects replication of data
  - In cache memory hierarchy or via SW replication and consistency

Naming Address Spaces

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

Three Fundamental Issues

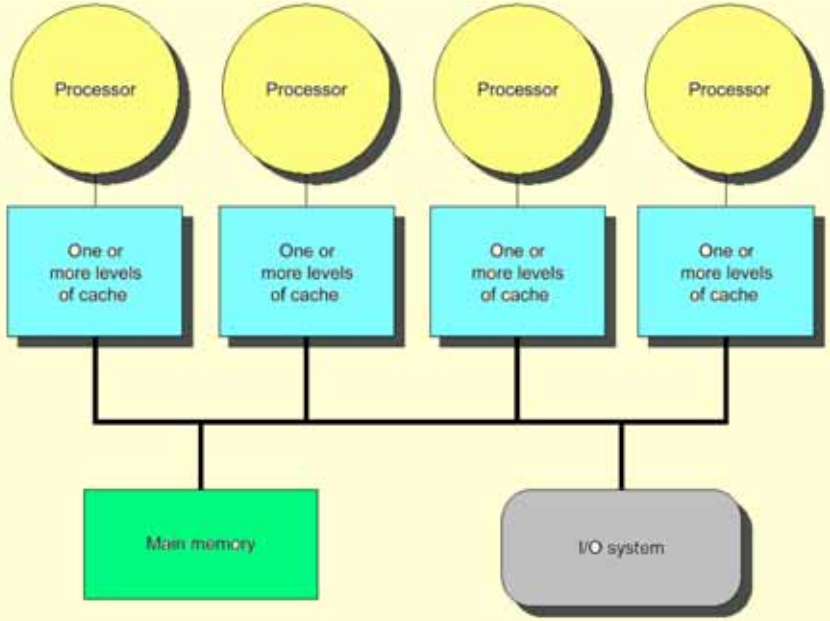
- 2: Synchronization: To cooperate, processes must coordinate
  - Message passing is implicit coordination with transmission or arrival of data
  - Shared address → additional operations to explicitly coordinate:
    e.g., write a flag, awaken a thread, interrupt a processor

Three Fundamental Issues

- 3: Latency and Bandwidth
  - Bandwidth
    - Need high bandwidth in communication
    - Cannot scale, but stay close
    - Match limits in network, memory, and processor
    - Overhead to communicate is a problem in many machines
  - Latency
    - Affects performance, since processor may have to wait
    - Affects ease of programming, since requires more thought to overlap communication and computation
  - Latency Hiding
    - How can a mechanism help hide latency?
    - Examples: overlap message send with computation, pre-fetch data, switch to other tasks

Centralized Shared Memory MIMD

- Processors share a single centralized memory through a bus interconnect
  - Memory contention: Feasible for small # processors
  - Caches serve to:
    - Increase bandwidth versus bus/memory
    - Reduce latency of access
    - Valuable for both private data and shared data
  - Access to shared data is optimized by replication
    - Decreases latency
    - Increases memory bandwidth
    - Reduces contention
    - Reduces cache coherence problems



Cache Coherency

A cache coherence problem arises when the cache reflects a view of memory which is different from reality

Time	Event	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
0				1
1	CPU A reads X	1		1
2	CPU B reads X	1	1	1
3	CPU A stores 0 into X	0	1	0

- A memory system is coherent if:
  - P reads X, P writes X, no other processor writes X, P reads X
    - Always returns value written by P
  - P reads X, Q writes X, P reads X
    - Returns value written by Q (provided sufficient W/R separation)
  - P writes X, Q writes X
    - Seen in the same order by all processors

Potential HW Coherency Solutions

- Snooping Solution (Snoopy Bus)
  - Send all requests for 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)

Potential HW Coherency Solutions

- Directory-Based Schemes
  - Keep track of what is being shared in one centralized place
  - 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 Snooping Protocols

- Write Invalidate Protocol:
  - Write to shared data: an invalidate is sent to all caches which snoop and invalidate any copies
  - Cache invalidation will force a cache miss when accessing the modified shared item
  - For multiple writers only one will win the race ensuring serialization of the write operations
  - Read Miss:
    - Write-through: memory is always up-to-date
    - Write-back: snoop in caches to find most recent copy

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

Basic Snooping Protocols

- Write Broadcast (Update) Protocol (typically write through):
  - Write to shared data: broadcast on bus, processors snoop, and update any copies
  - To limit impact on bandwidth, track data sharing to avoid unnecessary broadcast of written data that is not shared
  - Read miss: memory is always up-to-date
  - Write serialization: bus serializes requests!

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

Invalidate vs. Update

- Write-invalidate has emerged as the winner for the vast majority of designs
- Qualitative Performance Differences :
  - Spatial locality
    - WI: 1 transaction/cache block;
    - WU: 1 broadcast/word
  - Latency
    - WU: lower write–read latency
    - WI: must reload new value to cache

Invalidate vs. Update

- Because the bus and memory bandwidth is usually in demand, write-invalidate protocols are very popular
- Write-update can causes problems for some memory consistency models, reducing the potential performance gain it could bring
- The high demand for bandwidth in write-update limits its scalability for large number of processors