COE 308
Multiprocessors

Flynn's Tahonomy (1972)

- SISD (*Single Instruction Single Data*)
  - uniprocessors
- MISD (*Multiple Instruction Single Data*)
  - multiple processors on a single data stream;
- SIMD (*Single Instruction Multiple Data*)
  - same instruction is executed by multiple processors using different data
  - Adv.: simple programming model, low overhead, flexibility, all custom integrated circuits
  - Examples: Illiac-IV, CM-2
- MIMD (*Multiple Instruction Multiple Data*)
  - each processor fetches its own instructions and operates on its own data
  - Examples: Sun Enterprise 5000, Cray T3D, SGI Origin
  - Adv.: flexible, use off-the-shelf micros
  - MIMD current winner (< 128 processor MIMD machines)

Parallel Computers

- Definition: "A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast."

  Almasi and Gottlieb, Highly Parallel Computing ,1989

- Questions about parallel computers:
  - How large a collection?
  - How powerful are processing elements?
  - How do they cooperate and communicate?
  - How are data transmitted?
  - What type of interconnection?
  - What are HW and SW primitives for programmer?
  - Does it translate into performance?

Why Multiprocessors?

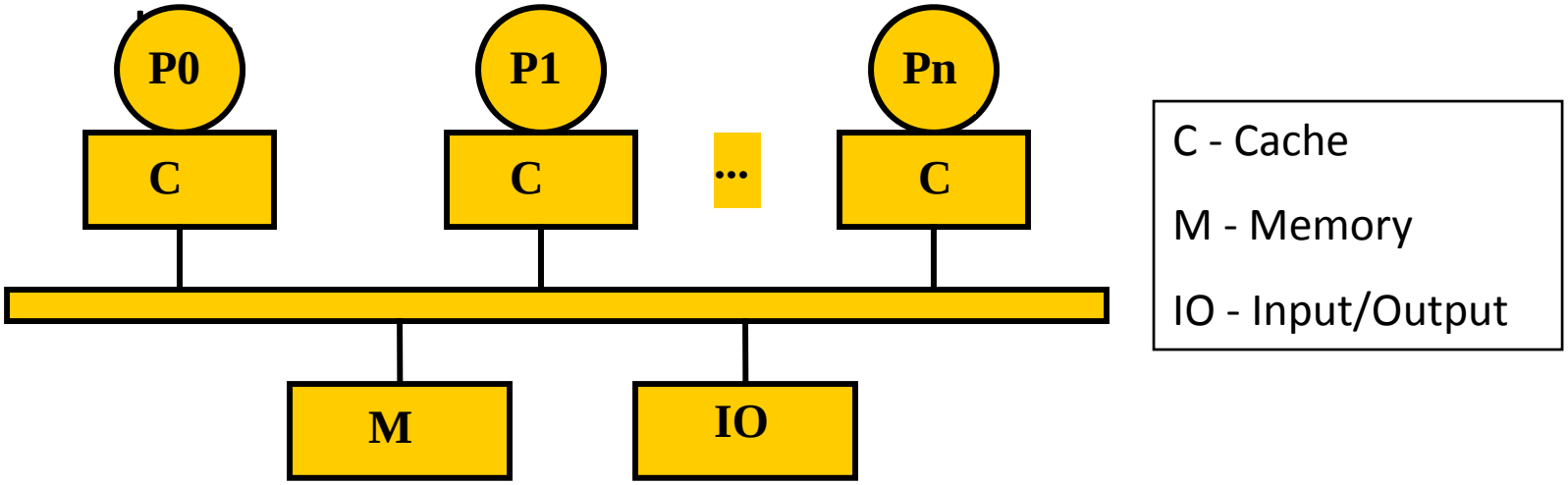
- Collect multiple microprocessors together
  to improve performance beyond a single processor
  - Collecting several more effective than designing a custom processor
- Complexity of current microprocessors
  - Do we have enough ideas to sustain 1.5X/yr?
  - Can we deliver such complexity on schedule?
- Slow (but steady) improvement in parallel software (scientific apps, databases, OS)
- Emergence of embedded and server markets driving microprocessors in addition to desktops
  - Embedded functional parallelism, producer/consumer model
  - Server figure of merit is tasks per hour vs. latency

MIMD

- Why is it the choice for general-purpose multiprocessors
  - Flexible
    - can function as single-user machines focusing on high-performance for one application,
    - multiprogrammed machine running many tasks simultaneously, or
    - some combination of these two
  - Cost-effective: use off-the-shelf processors
- Major MIMD Styles
  - Centralized shared memory
    ("Uniform Memory Access" time or "Shared Memory Processor")
  - Decentralized memory (memory module with CPU)

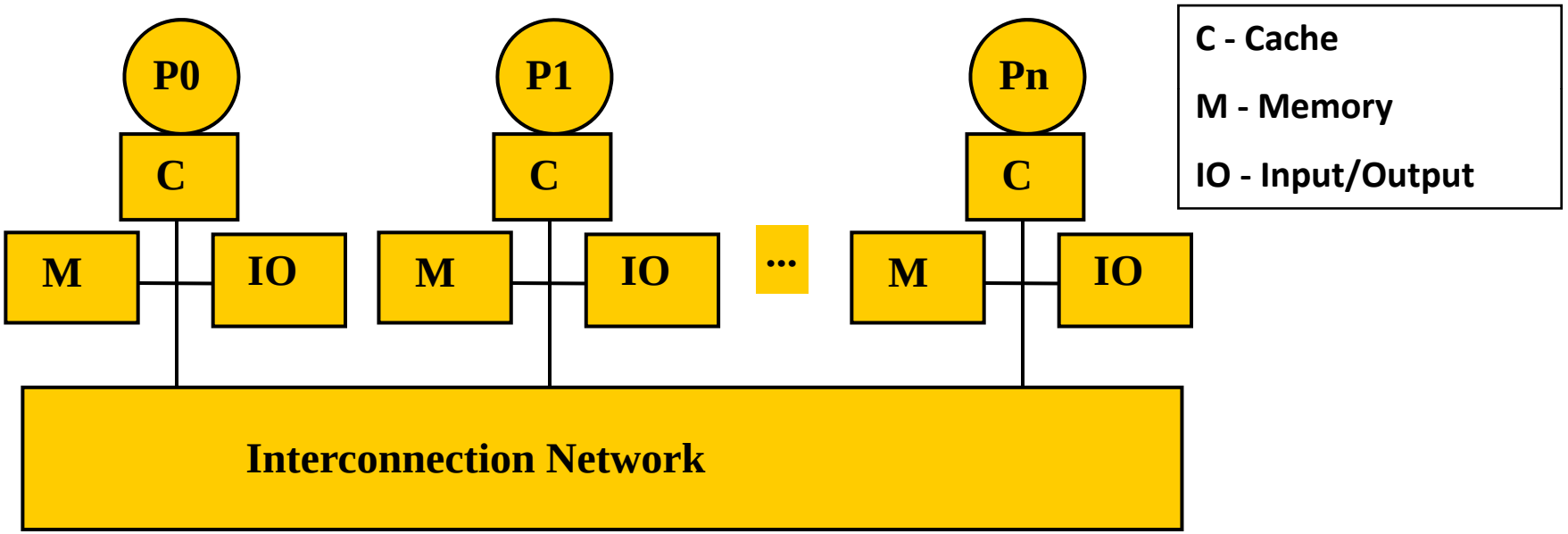
Centralized Shared-Memory Architecture

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



Distributed Memory Machines

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



Pro: Cost effective approach to scale memory bandwidth

Pro: Reduce latency for accesses to local memory

Con: Communication complexity

Memory Architectures

- DSM (Distributed Shared Memory)
  - physically separate memories can be addressed as one logically shared address space
    - the same physical address on two different processors refers to the same location in memory
- Multicomputer
  - the address space consists of multiple private address spaces that are logically disjoint and cannot be addressed by a remote processor
    - the same physical address on two different processors refers to two different locations in two different memories

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 SW model on either HW base

Amdahl's Law and Parallel Computers

- Amdahl's Law (FracX: original % to be speed up)
  Speedup = 1 / [(FracX/SpeedupX + (1-FracX)]
- A portion is sequential => limits parallel speedup
  - Speedup <= 1/ (1-FracX)
- Ex. What fraction sequential to get 80X speedup from 100 processors? Assume either 1 processor or 100 fully used
- 80 = 1 / [(FracX/100 + (1-FracX)]
- 0.8*FracX + 80*(1-FracX) = 80 - 79.2*FracX = 1
- FracX = (80-1)/79.2 = 0.9975
- Only 0.25% sequential!

Performance Metrics: Latency and Bandwidth

- Bandwidth
  - Need high bandwidth in communication
  - Match limits in network, memory, and processor
  - Challenge is link speed of network interface vs. bisection bandwidth of network
- Latency
  - Affects performance, since processor may have to wait
  - Affects ease of programming, since requires more thought to overlap communication and computation
  - Overhead to communicate is a problem in many machines
- Latency Hiding
  - How can a mechanism help hide latency?
  - Increases programming system burden
  - Examples: overlap message send with computation, prefetch data, switch to other tasks

Shared Address Model Summary

- Each processor can name
  every physical location in the machine
- Each process can name
  all data it shares with other processes
- Data transfer via load and store
- Data size: byte, word, ... or cache blocks
- Uses virtual memory to map
  virtual to local or remote physical
- Memory hierarchy model applies:
  now communication moves data to local processor cache (as load moves data from memory to cache)
  - Latency, BW, scalability when communicate?

Shared Address/Memory Multiprocessor Model

- Communicate via Load and Store
  - Oldest and most popular model
- Based on timesharing: processes on multiple processors vs. sharing single processor
- Process: a virtual address space and ~ 1 thread of control
  - Multiple processes can overlap (share), but ALL threads share a process address space
- Writes to shared address space by one thread are visible to reads of other threads
  - Usual model: share code, private stack, some shared heap, some private heap

SMP Interconnect

- Processors to Memory AND to I/O
- Bus based: all memory locations equal access time so SMP = “Symmetric MP”
  - Sharing limited BW as add processors, I/O

Message Passing Model

- Whole computers (CPU, memory, I/O devices) communicate as explicit I/O operations
  - Essentially NUMA but integrated at I/O devices vs. memory system
- Send specifies local buffer + receiving process on remote computer
- Receive specifies sending process on remote computer + local buffer to place data
  - Usually send includes process tag
    and receive has rule on tag: match 1, match any
  - Synch: when send completes, when buffer free, when request accepted, receive wait for send
- Send+receive => memory-memory copy, where each each supplies local address, AND does pairwise sychronization!

Advantages of Shared-Memory Communication Model

- Compatibility with SMP hardware
- Ease of programming when communication patterns are complex or vary dynamically during execution
- Ability to develop apps using familiar SMP model,
  attention only on performance critical accesses
- Lower communication overhead, better use of BW for small items, due to implicit communication and memory mapping to implement protection in hardware, rather than through I/O system
- HW-controlled caching to reduce remote comm.
  by caching of all data, both shared and private

Advantages of Message-passing Communication Model

- The hardware can be simpler (esp. vs. NUMA)
- Communication explicit => simpler to understand; in shared memory it can be hard to know when communicating and when not, and how costly it is
- Explicit communication focuses attention on costly aspect of parallel computation, sometimes leading to improved structure in multiprocessor program
- Synchronization is naturally associated with sending messages, reducing the possibility for errors introduced by incorrect synchronization
- Easier to use sender-initiated communication,
  which may have some advantages in performance