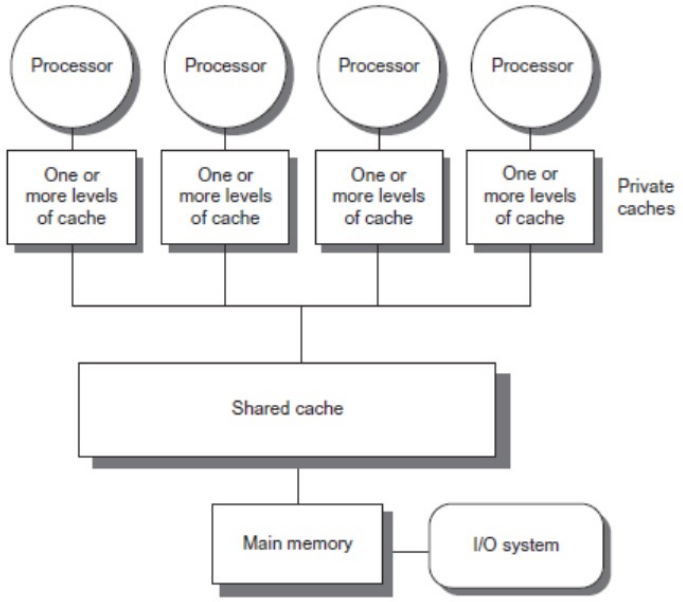
Introduction

- Thread-Level parallelism
  - Have multiple program counters
  - Uses MIMD model
  - Targeted for tightly-coupled shared-memory multiprocessors
- For *n* processors, need *n* threads
- Amount of computation assigned to each thread = grain size
  - Threads can be used for data-level parallelism, but the overheads may outweigh the benefit

Architectures

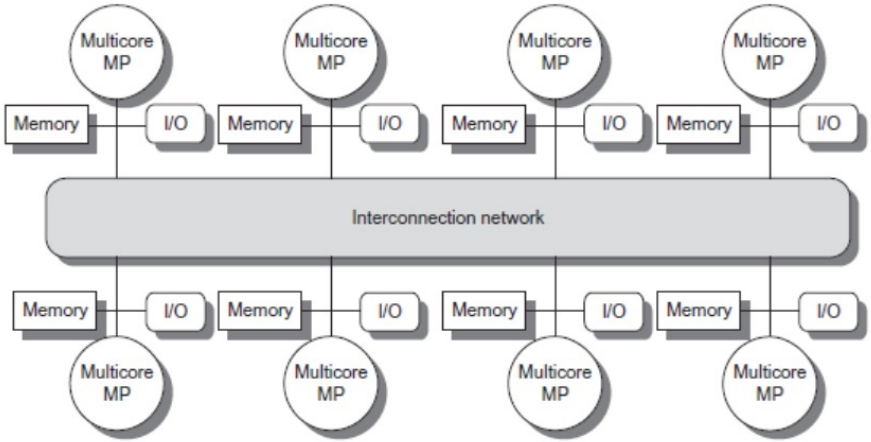
Symmetric multiprocessors (SMP)

- Small number of cores
- Share single memory with uniform memory latency
- *Uses Snoopy Cache Protocols*



Distributed shared memory (DSM)

- Memory distributed among processors
- Non-uniform memory access/latency (NUMA)
- Processors connected via dire (switched) and non-direct (multi-hop) interconnection networks
- *Uses Directory Cache Protoc*



Cache Coherence

- Coherence
  - All reads by any processor must return the most recently written value
  - Writes to the same location by any two processors are seen in the same order by all processors
- Consistency
  - When a written value will be returned by a read
  - If a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

Enforcing Coherence

- Coherent caches provide:
  - *Migration*: movement of data
  - *Replication*: multiple copies of data
- Cache coherence protocols
  - Directory based
    - Sharing status of each block kept in one location
  - Snooping
    - Each core tracks sharing status of each block

Snoopy Cache Protocols

- Write invalidate
  - On write, invalidate all other copies
  - Use bus itself to serialize
    - Write cannot complete until bus access is obtained

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

- Write update
  - On write, update all copies

Snoopy Coherence Protocols

- Locating an item when a read miss occurs
  - In write-back cache, the updated value must be sent to the requesting processor
- Cache lines marked as shared or exclusive/modified <- *Caution: "modified" takes on different meanings for different protocols*
  - Only writes to shared lines need an invalidate broadcast
    - After this, the line is marked as exclusive

Common Snoopy Protocols

- MSI: Modified, Shared, Invalid
- MESI: Modified, +(Exclusive), Shared, Invalid
- MOESI: Mod, +(Owned), Exc. Shared, Invalid

Coherence Protocols

- Every multicore with >8 processors uses an interconnect other than bus
  - Makes it difficult to serialize events
  - Write and upgrade misses are not atomic
  - How can the processor know when all invalidates are complete?
  - How can we resolve races when two processors write at the same time?
  - Solution: associate each block with a single bus

Directory Based Protocols

- Snooping schemes require communication among all caches on every cache miss
  - Limits scalability
  - Another approach: Use centralized directory to keep track of every block
    - Which caches have each block
    - Dirty status of each block
- Implement in shared L3 cache
  - Keep bit vector of size = # cores for each block in L3
  - Not scalable beyond shared L3