ECE 2300
Digital Logic & Computer Organization

Fall 2016

More Advanced Topics
Case Study

Cornell University

Parallelism: Making our Processor *Fast*

- Processor architects improve performance through hardware that exploits the different types of parallelism within computer programs

- Instruction-Level Parallelism (ILP)
  - Parallelism within a sequential program
- Thread-level parallelism (TLP)
  - Parallelism among different threads
- Data-level parallelism (DLP)
  - Parallelism among the data within a sequential program

Multiprocessing and Multicore

- ***Multiprocessor***: Computer system with >1 CPU
  - Computer with 1 CPU is called a *uniprocessor*

- ***Multicore***: Multiprocessor on a single chip

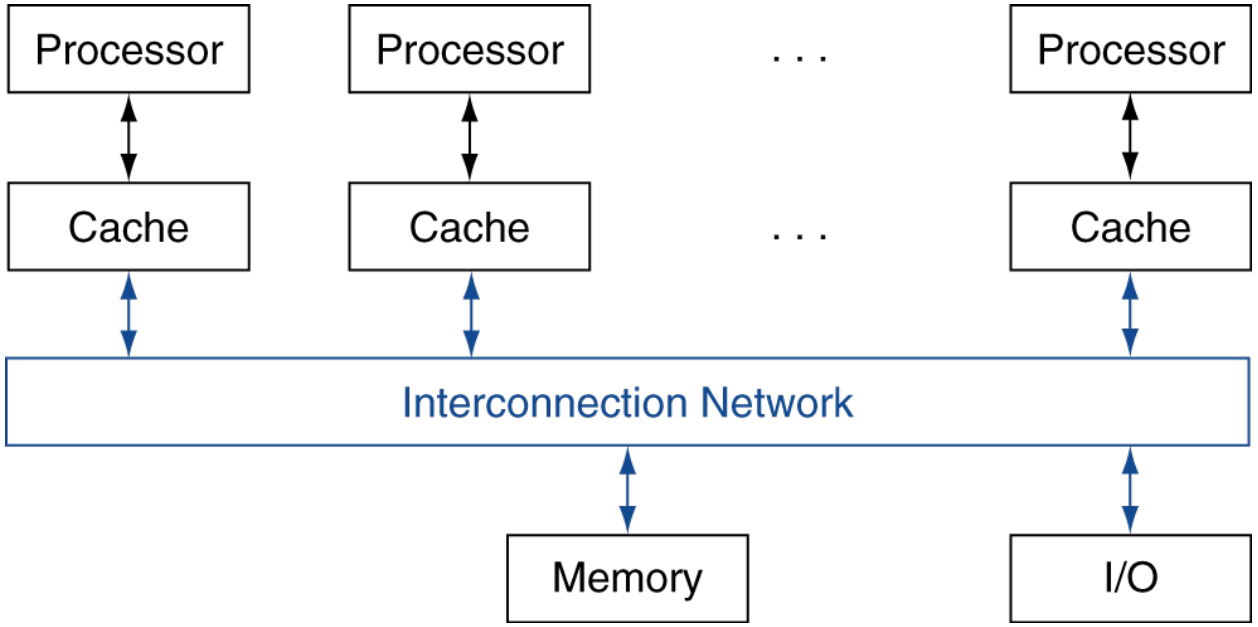
Multiprocessor Advantages

- **Performance improvement by splitting task among multiple CPUs (thread-level parallelism)**
  - **Scientific programs, databases, media processing, web browsers, games, etc**
- **Tolerance to failures**
  - **If one CPU fails, the system is still usable**
- **Power efficiency**
  - **Several modestly superscalar CPUs can be more power efficient than 1 wide CPU**
  - **Major reason for the move to multicore chips**

Multiprocessor Challenges

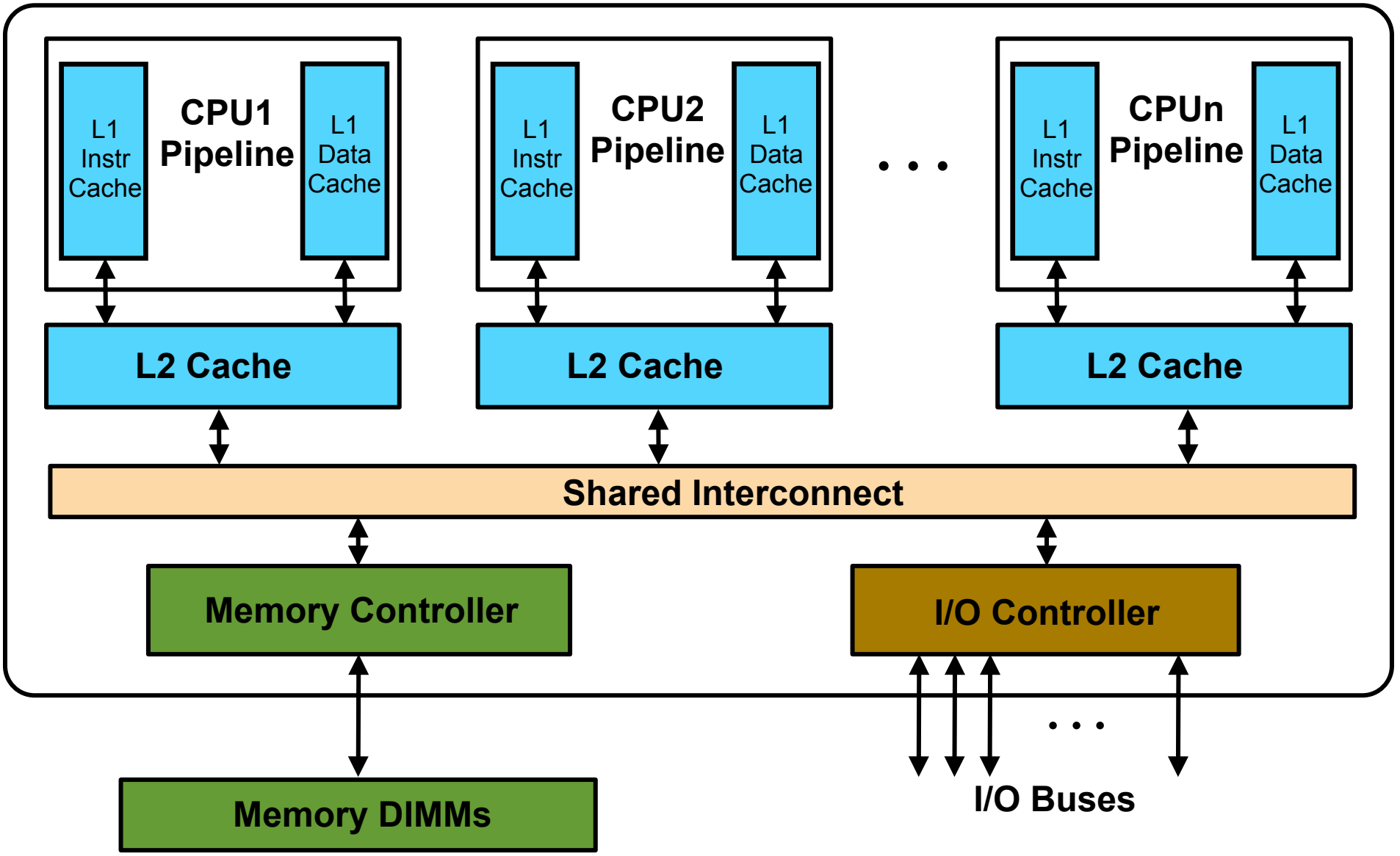
- Parallel programming is difficult and error prone
- Some programs are not easy to parallelize
- The serial part of a parallel program limits the performance improvement (Amdahl's Law)
- Performance cost of communication and coordination among threads

Shared Memory Multiprocessor



- All CPUs share same memory and I/O devices
- CPUs share data by writing/reading memory
- Widely used in multicore chips

Typical Multicore Chip Organization



Cache Coherence Problem

- Multiple copies of same data may exist in different CPU caches and in memory
- Write by one CPU to its cache makes other copies "stale" (out of date)
  - The caches are no longer "coherent"
- Example
  - CPU 1 reads block A from memory into cache
  - CPU 2 reads block A from memory into cache
  - CPU 1 writes to block A in its cache
  - CPU 2 and memory now hold stale copies of block A

Cache Coherence Solution

- Each block in cache has *state* bits that indicate the *ownership* of the block
- Before writing to a block, cache controller *invalidates* (gets rid of) other cache copies to obtain an *exclusively owned* copy
- All caches *snoop* (monitor) the shared interconnect to determine when they should
  - Invalidate their copy of a block when another cache asks for exclusive ownership
  - Provide their copy of a block (instead of memory) to a requester

MESI Cache Coherence Solution

- Each block in the cache is in one of 4 states
  - *Invalid*: Block is not valid.
  - *Shared*: Another cache may have the block. Copy in memory is the same as in cache.
  - *Exclusive*: No other cache has the block. Copy in memory is the same as in cache.
  - *Modified*: No other cache has the block. Copy in memory is stale (out of date).

- A cache obtains an exclusive copy of a block before writing to it
- A cache with a modified copy of a block provides it upon request (instead of memory)

MESI Cache Coherence Example

- **Previous problem**
  - CPU 1 reads block A from memory into cache
  - CPU 2 reads block A from memory into cache
  - CPU 1 writes to block A in its cache
  - CPU 2 and memory now hold stale copies of block A

- **MESI solution**
  - Before CPU 1 does its write, it first broadcasts an *invalidate A* message on the shared interconnect
  - CPU 2 marks its block A as *invalid*
  - CPU 1 does its write and marks its block A *modified*
  - CPU 2 will now miss when it wants to read A again
  - CPU 1 provides its copy of A instead of memory