Why CMPs?
- It is now possible to implement a single-chip multiprocessor in the same area as a wide issue superscalar
- **Motivation**
  - Area and Power has been increasing at rates greater than single-thread performance
  - Under-utilized superscalar execution resources
    - Branch mispredictions
    - Data dependences
    - Cache misses

Comparing CMPs to Superscalar Processors
- Performance comparison depends on application
- **Wall’s classification**
  - Applications with low to moderate parallelism
    - IPC < 10
    - Mostly integer applications
  - Applications with large amount of parallelism
    - IPC > 40
    - Mostly floating point applications

Two Microarchitectures: 6-way SS vs. 4 2-way CMP
- For fair comparison, we need almost equal areas (die sizes) for SS and CMP
- Superscalar: 6-way R10000-like machine
  - More ports, bigger structures compared to the 4-way R10000
- CMP: 4 cores, each 2-way superscalar
  - Each core similar to the Alpha 21064 (1992)
  - Shared L2 cache
  - Clock speed?
- **Design parameters:** Paper Table 1
- **Floor plans:** Paper Figures 2 and 3
- **Results:** Paper tables 4-7, figures 4-6

Discussion: CMP Issues
- CMP or Superscalar?
  - Low parallelism favors superscalar
  - Medium parallelism about even
  - High parallelism favors CMP
- For CMPs, focus is less on core and more on “uncore”
  - Cache hierarchy and organization
  - Interconnection network
  - Bus interface
  - Increased demand for bus bandwidth
  - Cache miss latency
  - Programmability
  - Amdahl’s law

Hyper-Threading Technology
- Makes single physical processor appear as two logical processors
  - Execution resources and caches are shared
  - Architectural state is duplicated
- **Motivation**
  - Area and Power have been increasing at rates greater than single-thread performance (Marr et al., figure 1)
  - Under-utilized superscalar execution resources
    - Branch mispredictions
    - Data dependences
    - Cache misses
Thread-Level Parallelism (TLP)
- Many software applications consist of multiple threads or processes
  - From same application
  - From different applications
  - From operating system services
  - From OS threads doing background maintenance
- How to exploit TLP?
  - Traditional multi-chip multiprocessors
  - Single chip multiprocessing (CMP)
  - Multithreading
  - Wider pipelines in superscalar processors

CMP vs Multithreading
- CMP has full set of resources per logical processor
  - Execution time more predictable
  - Makes scheduling easier
- Multithreading gives best power and area efficiency
  - Better resource utilization
  - Fewer processor utilization
  - Less static power
- Marr et al., figures 2 and 3

Intel Hyper-Threading Goals
- Minimize die area cost of implementation
- Ensure when one logical processor stalls, the other logical processor could make forward progress
- Ensure a single logical process runs at the same speed on a hyper-threaded processor as it would run on the exact same processor without hyper-threading support
- Some microarchitecture changes are necessary to implement hyper-threading
  - Marr et al., figures 4, 5, and 6

How to Evaluate Multiprocessor Performance
- To compare the performance of a program P running on two different systems A and B, the speedup of A over B is:
  $$\text{Speedup}(A) = \frac{\text{Time/Program}(B)}{\text{Time/Program}(A)}$$
- Iron Law:
  $$\text{Time/Program} = \text{Instruction/Program} \times \text{Cycles/Instruction} \times \text{Time/Cycle}$$
- Usually, Instructions/Program and Time/Cycle are fixed for single-threaded applications
- Speedup is estimated using CPI or IPC only

Why Can IPC be Misleading?
- Instructions/Program is not fixed
- Operating system’s scheduling decisions can cause widely divergent executions
  - Idle time
  - Spin-lock wait time
  - Privileged code (e.g., TLB miss handler)
- The same amount of useful work can be done even though number of instructions between different executions may vary

How Can IPC be Misleading?
- Worst case scenario: Using IPC leads to opposite conclusion
- Other scenarios:
  - IPC overestimates speedup
  - IPC underestimates speedup
  - IPC’s results are inconclusive
- Figures 1-4 in the paper show different experiments
- Some incomplete solutions discussed in paper
  - Ignoring system code
  - Ignoring spin locks
  - Trace-driven simulation
- Solution: Use runtime or other work-related metrics to evaluate multithreaded applications
What is a Shared Memory Architecture?
- All processors can access all memory
- Processors share memory resources, but can operate independently
- One processor's memory changes are seen by all other processors
- Easier to program
  - Communication through shared memory
  - Synchronization through locks stored in shared memory
- Need cache coherence in hardware
  - Why?
- Need interconnection network between all processors and all memory

Shared Memory Architectures
- Uniform Memory Access (UMA) Architecture
  - Example: Symmetric Multiprocessor (SMP) machines
  - Identical processors with equal access and equal access time to memory
  - Also called CC-UMA - Cache Coherent UMA. Cache coherence means if one processor updates a location in shared memory, all the other processors know about the update
- Non-Uniform Memory Access (NUMA) Architecture
  - Often made by physically linking two or more SMPs
  - One processor can directly access memory of another processor
  - Not all processors have equal access time to all memories
  - Memory access across link is slower
  - Called CC-NUMA if Cache Coherence is maintained

Shared Bus Architectures
- Contention for bus and memory may degrade performance
- Need arbitration for the bus (whenever more than one bus master exists)
- Some (old) examples for shared bus architectures:
  - Encore’s Multimax: Paper figure 1
  - Sequent Balance: Paper figure 2
  - Alliant FX/80: Paper figure 3
  - ELXSI System 6400: Paper figure 4

Network Multiprocessors
- More scalable than shared bus architectures
  - Less contention for shared interconnection resources
  - Usually higher latency to communicate
  - May need arbitration to access shared memory (if more than one processor requests access to same bank)
- Some (old) examples:
  - BBN Butterfly: Paper figure 5, 6
  - Intel iPSC/2
  - NCUBE/n
  - FPS T Series

Interconnection Networks
- In a shared memory MP, we need to connect different processors and memory modules
- Types of interconnect:
  - Shared bus
  - Crossbar: Fully connected
  - Ring
  - Mesh
  - 2-D Torus
  - Hypercube
- Number of hops vs. number of links: Compare N processors and M memory modules
- More details later in the course
Memory Hierarchy

- Problem: sharing memory means more than one processor can send requests to memory
  - High memory bandwidth required
- To avoid sending lots of memory requests, processors use caches to:
  - Filter out many memory requests
  - Reduce average memory latency
  - Reduce memory bandwidth requirements
- Typically more than one level of caches is used
  - L1 caches: Usually Split I & D caches, small and fast
  - L2 caches: Usually on die, composed of SRAM cells
  - L3 caches: On-die or off-die, SRAM or eDRAM cells

Cache Coherence

- Problem: Using caches means multiple copies of the same memory location may exist
  - Updates to the same location may lead to bugs
- Example:
  - Processor 1 reads A
  - Processor 2 reads A
  - Processor 1 writes to A
  - Now, processor 2’s cache contains stale data
  - Cache coherence need to be implemented in hardware using a cache coherence protocol

Conditions for Cache Coherence

- Program Order. A read by processor P to location A that follows a write by P to A, with no writes to A by another processor in between, should always return the value of A written by P
- Coherent View of Memory. A read by processor P1 to location A that follows a write by another processor P2 to location A should return the written value by P2 if:
  - The read and write are sufficiently separated in time
  - No other writes to A by another processor occur between the read and the write
- Write Serialization. Writes to the same location are serialized: Two writes to the same location by any two processors are seen in the same order by all processors

Cache Coherence (Cont.)

- Cache coherence defines behavior of reads and writes to the same memory location
- Memory consistency models define the behavior of reads and writes with respect to accesses to other memory locations (More details later in the course)
- Two main types of cache coherence protocols:
  - Snooping
    - Caches keep track of the sharing status of all blocks
    - No centralized state is kept
    - Cache controllers snoop shared interconnect to know when a requested block exists in the cache
  - Directory
    - Sharing status of any block in memory is kept in one location

Very Simple Coherence Protocol

- MI protocol
  - Two states: M (Modified) and I (Invalid)
  - Only one cache contains a copy of a certain memory location
  - When another cache requests a block, the cache currently containing the block invalidates it
  - Protocol limits sharing and degrades performance

- Optimization: MSI protocol allows read sharing

Reading Assignment

- Homework 2 out today, due on Oct 14
- Project proposals due Thursday