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 stalls
  - 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:
  \[ \text{Speedup}(A) = \frac{\text{Time}/\text{Program}(B)}{\text{Time}/\text{Program}(A)} \]
- Iron Law:
  \[ \text{Time}/\text{Program} = \text{Instruction}/\text{Program} \]
  \* Cycles/Instruction
  \* Time/Cycle
- Usually, Instructions/Program and Time/Cycle are fixed for single-threaded applications
  - Speedup is estimated using CPI or IPC only
- 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

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
Reading Assignment

- **Thursday:**
- Homework 2 out today, due on Jan 19