Hyper-Threading Technology

- Makes single physical processor appear as two logical processors
  - Execution resources and caches are shared
  - Architecture State is duplicated
- Motivation
  - Area and Power have been increasing at rates greater than single-thread performance (Paper 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 operating system threads doing background maintenance
- Examples?

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 are efficiency
  - Better resource utilization
  - Less processor stalls
  - Less static power
- Paper figures 2 and 3

Types of Multithreading

- Time-slice multithreading
- Switch-on-event multithreading
- Fine-grain multithreading
- Simultaneous multithreading

- Discuss pros and cons
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

Hyper-Threading Design and Performance

- Some microarchitecture changes are necessary to implement hyper-threading
  - Paper figures 4, 5, and 6
- Hyper-threading improves performance for some application classes
  - Paper figures 8 and 9

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:
  \[ \frac{\text{Time}}{\text{Program}} = \frac{\text{Instruction}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}} \]
  - 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

Reading Assignment

- No class on Monday, Jan 19
- Wednesday:
  - Homework 2 out today, due on Jan 28.