Contents

Thread-Level Parallelism

- Instruction level parallelism exploits very fine grain independent instructions
- Thread level parallelism is explicitly represented in the program by the use of multiple threads of execution that are inherently parallel
- Goal: Use multiple instruction streams to improve either
  1. Throughput of computers that run many programs
  2. Execution time of multi-threaded programs
- Thread level parallelism potential allows huge speedups
- There is no complex superscalar architecture that scales poorly
- There is no particular requirement for very complex compilers, like for VLIW
- Instead the burden of identifying and exploiting the parallelism falls mostly on the programmer
  - Programming with multiple threads is much more difficult than sequential programming
  - Debugging parallel programs is incredibly challenging
- But it's pretty easy to build a big, fast parallel computer
  - However, the main reason we have not had widely-used parallel computers in the past is that they are too difficult (expensive), time consuming (expensive) and error prone to program

Multithreading in a Processor Core

- Find a way to "hide" true data dependency stalls, cache miss stalls, and branch stalls by finding instructions (from other process threads) that are *independent* of those stalling instructions

- Multithreading - increase the utilisation of resources on a chip by allowing multiple processes (*threads*) to share the functional units of a single processor
  - Processor must duplicate the state hardware for each thread - a separate register file, PC, instruction buffer, and store buffer for each thread
  - The caches, TLBs, branch predictors can be shared (although the miss rates may increase if they are not sized accordingly)
  - The memory can be shared through virtual memory mechanisms
  - Hardware must support efficient thread context switching

Types of Multithreading

- *Fine-grain* - switch threads on every instruction issue
  - Round-robin thread interleaving (skipping stalled threads)
  - Processor must be able to switch threads on every clock cycle
  - Advantage - can hide throughput losses that come from both short and long stalls
  - Disadvantages - slows down the execution of an individual thread since a thread that is ready to execute without stalls is delayed by instructions from other threads
- *Course-grain* - switches threads only on costly stalled (e.g. L2 cache misses)
  - Advantages - thread switching doesn't have to be essentially free and must less likely to slow down the execution of an individual thread
  - Disadvantage - limited, due to pipeline start-up costs, in its ability to overcome throughput loss
    * Pipeline must be flushed and refilled on thread switches

Simultaneous Multithreading (SMT)

- A variation on multithreading that uses the resources of a multiple-issue, dynamically scheduled processor (superscalar) to exploit both program ILP and *thread-level parallelism* (TLP)
  - Most superscalar processors have more machine level parallelism than more programs can effectively use (i.e. than have ILP)
  - With register renaming and dynamic scheduling, multiple instructions from independent threads can be issued without regard to dependencies among them
    * Need separate rename tables (reorder buffers) for each thread

* Need the capability to commit from multiple (i.e., from multiple recorder buffers) in one cycle