DEPARTMENT OF
INFORMATION SYSTEMS
AND COMPUTER SCIENCE

Parallel Processing

Working at the Same Time

>

Working One at a Time

Learning Objectives

- Overview: Techniques for Speeding Up CPUs
- Applications of Parallel Processing
- Parallel Processing on a Wider Scale

Moore's Law

- Machines are getting faster:
  - Every 12-24 months, you can get a processor with twice the speed for the same cost.
  - However, they're ~~never~~ still not fast enough!
- These computationally-intensive apps can take days, even weeks, on the latest processors!
  - Physics: Computational fluid dynamics (CFD), weather, etc.
  - Bio/Chem: Modeling of DNA, proteins, etc.
  - Business: Data mining, simulations, predictions, etc.
  - Multimedia: CGI rendering, games, simulations, etc.
  - Math: Finding large prime numbers, cryptography, etc.

Speeding Up CPUs

- More aggressive pipelining
  - Super-Pipelining
  - Out-of-order Execution
  - Branch Prediction
- Instruction-Level Parallelism
  - Superscalar Processors
  - Very Long Instruction Word
- Thread- / Process-Level Parallelism
  - Hyperthreading
  - Symmetric Multiprocessors
- Explicit Parallelism
  - Single Instruction, Multiple Data
  - Multiple Instruction, Multiple Data
  - Symmetric Multiprocessors (again)
- Distributed Parallelism
  - Clusters
  - Grid Computing

Super-Pipelining

- More than 5 stages.
- The ALU and data memory parts are usually the ones pipelined into more stages.
- Lower minimum clock period (higher frequency = higher MIPS) but potentially more stalls (higher CPI = lower MIPS)
- Super-pipelining is used in conjunction with other techniques to maximize its effectiveness.

Out-of-Order Execution

- In the IF stage, look at several instructions ahead of time.
- If one of them needs to be stalled, look for other instructions after it that can be run already.
  - Hardware is reordering the instructions instead of inserting NOPs!
  - Since it is done by the hardware, there is still no need to keep different versions of the code – the unpipelined version is good enough.

Branch Prediction

- Predict the most-probable branch target in the IF stage instead of just blindly pre-fetching the next (PC+4) instruction.
- If prediction is (hopefully most of the time) correct, don't annul. Otherwise (hopefully rarely), annul.
- Can base prediction on code behavior – branches that go backward are taken more often than not since they usually belong to loops.
- Can use Branch-Target-Buffer (BTB)
  - Cache result of a branch instruction and use that as predicted target if we run into that branch instruction again.
- Can assist in selecting which instructions should be considered for out-of-order execution.

Superscalar

- Use several pipelines and assign “ready” (no need to stall) instructions to available pipelines.
- Technically still a single CPU but there are multiple instances of certain units (such as the ALU and Floating-Point Unit / FPU).
  - Remember the two dryers in the laundromat?
- Now more than one instruction can be finished in a cycle → CPI becomes less than 1 → Higher MIPS.
- Code is still sequential.

Very Long Instruction Word

- Code is written as groups of instructions that can be run in parallel.
- Parallelism is determined by the compiler.
  - Resulting code is not compatible with non-VLIW versions.

Dataflow Parallelism

- Out-of-order execution, superscalar processors, and VLIW take advantage of <u>independent</u> instructions in the code.
  - These are the instructions that do NOT need to stall at the time.
- How do we find/determine these independent instructions?
  - How do we know when NOT to stall?
  - How do we know what instructions to move when reordering the code?
    - ~~Why does this slide set have no pictures?!?~~

Symmetric Multiprocessors

- Multiple, independent CPUs.
  - Multiple-processor servers (system level).
  - Or Intel Core 2's quad-core model (multi-chip).
  - Or Intel Core 2's dual-core model (single-chip).
- Shared memory but processors have their own caches (works better with thread- / process-level parallelism).
- Processors run independently.
  - Can run completely different programs or threads.
  - Can run the same program on different data (explicit parallelism).

"Embarrassingly Parallel"

- The idea of parallel processing is to divide the problem into many parts, and then have different processors work on different parts at the same time.
- An "embarrassingly parallel" application is parallelism at its simplest.
  - Each part is independent.
  - Processors can work without communicating with / waiting for each other.
    - Except maybe to inform a "master" processor that they're done with their part.

Adding N Numbers

- **Example: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8**

- **Single Processor**

  1+2+3+4+5+6+7+8

  3+3+4+5+6+7+8

  6+4+5+6+7+8

  10+5+6+7+8

  15+6+7+8

  21+7+8

  28+8

  7 steps!

- **Dual Processor**

  1+2+3+4+5+6+7+8

  3+7+5+6+7+8

  10+11+7+8

  21+15 (idle)

  ***4 steps!* (1.75x faster!)**

- **In general, programs will be up to N times faster with N processors.**

DISCS

Parallel Processing - How?

- Some of you might already have computers with multiple processors (symmetric multiprocessors/SMPs), but still find them too slow.
- What you need: SUPERCOMPUTERS
  - VERY fast computers with MANY processors.
- Different types:
  - SIMD – Single-instruction, multiple data
  - MIMD – Multiple-instruction, multiple data (more common type)
    - Centralized Shared Memory (SMP) – simpler memory model but less scalable (up to 32 processors only)
    - Distributed Memory – much more scalable because each processor has its own memory.
- Drawback: $$$ + not easily upgradable

Distributed Parallel Processing

- Of course, your computing power needs do not have to be centralized in one place.
- Clusters
  - Several computers connected by a high-speed network.
  - Usually programmed using a Message Passing Interface (MPI).
- Grid Computing
  - Geographically distributed parallel processing.
  - This means slower networks but potentially a LOT more computers.
  - Can include clusters as grid nodes.

What Does the Future Hold?

- Grid computing allows users to:
  - Share and distribute computing resources and power across the Net – computing power might become a trade-able commodity.
  - Access said computational resources and power through a simple interface – computational web services “hide” the actual resources being used.
    - Like an electrical power grid, but for computing power.
    - Strangely enough, there's also something called power line communication (electricity + Internet).
- The future is... CLOUD~~y~~.