Performance

Kevin Walsh
CS 3410, Spring 2010
Computer Science
Cornell University

See: P&H 1.4
What to look for in a computer system?

- Correctness: negotiable?
- Cost
  - purchase cost = f(silicon size = gate count, economics)
  - operating cost = f(energy, cooling)
  - operating cost >= purchase cost
- Efficiency
  - power = f(transistor usage, voltage, wire size, clock rate, ...)
  - heat = f(power)
    - Intel Core i7 Bloomfield: 130 Watts
    - AMD Turion: 35 Watts
    - Intel Core 2 Solo: 5.5 Watts
- Performance
- Other: availability, size, greenness, features, ...
How to measure performance?

- GHz (billions of cycles per second)
- MIPS (millions of instructions per second)
- MFLOPS (millions of floating point operations per second)
- Benchmarks (SPEC, TPC, ...)

Metrics
- Latency: how long to finish my program
- Throughput: how much work finished per unit time
Assumptions:
- alu: 32 bit ripple carry + some muxes
- next PC: 30 bit ripple carry
- control: minimized for delay
- program memory: 16 ns
- register file: 2 ns access
- ignore wires, register setup time
- transistors: 2 ns per gate

Better Still:
- next PC: cheapest adder faster than 21 gate delays

Better:
- alu: 32 bit carry lookahead + some muxes
- next PC: 30 bit carry lookahead

All signals are stable
80 gates = 160 ns; 21 gates = 42 ns after clock edge
→ ~ 6 MHz; ~ 24 MHz;

Note! 1 light ns = 1 ft
<table>
<thead>
<tr>
<th>32 Bit Adder Design</th>
<th>Space</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ripple Carry</td>
<td>(\approx 300) gates</td>
<td>(\approx 64) gate delays</td>
</tr>
<tr>
<td>2-Way Carry-Skip</td>
<td>(\approx 360) gates</td>
<td>(\approx 35) gate delays</td>
</tr>
<tr>
<td>3-Way Carry-Skip</td>
<td>(\approx 500) gates</td>
<td>(\approx 22) gate delays</td>
</tr>
<tr>
<td>4-Way Carry-Skip</td>
<td>(\approx 600) gates</td>
<td>(\approx 18) gate delays</td>
</tr>
<tr>
<td>2-Way Look-Ahead</td>
<td>(\approx 550) gates</td>
<td>(\approx 16) gate delays</td>
</tr>
<tr>
<td>Split Look-Ahead</td>
<td>(\approx 800) gates</td>
<td>(\approx 10) gate delays</td>
</tr>
<tr>
<td>Full Look-Ahead</td>
<td>(\approx 1200) gates</td>
<td>(\approx 5) gate delays</td>
</tr>
</tbody>
</table>
Critical Path

- Longest path from a register output to a register input
- Determines minimum cycle, maximum clock frequency

Strategy 1

- Optimize for delay on the critical path
- Optimize for size / power / simplicity elsewhere
Strategy 2

- Multiple cycles to complete a single instruction

E.g: Assume:

- load/store: 100 ns
- arithmetic: 50 ns
- branches: 33 ns

Multi-Cycle CPU

30 MHz (33 ns cycle) with
- 3 cycles per load/store
- 2 cycles per arithmetic
- 1 cycle per branch

Faster than Single-Cycle CPU?

10 MHz (100 ns cycle) with
- 1 cycle per instruction
Instruction mix for some program P, assume:

- 25% load/store (3 cycles / instruction)
- 60% arithmetic (2 cycles / instruction)
- 15% branches (1 cycle / instruction)

Multi-Cycle performance for program P:

\[ 3 \times 0.25 + 2 \times 0.60 + 1 \times 0.15 = 2.1 \]

average cycles per instruction (CPI) = 2.1

Multi-Cycle @ 30 MHz
Single-Cycle @ 10 MHz
Single-Cycle @ 15 MHz

800 MHz PIII “faster” than 1 GHz P4
Goal:
Make P run 2x faster via faster arithmetic instructions

*Instruction mix (for P):*
- 25% load/store, CPI = 3
- 60% arithmetic, CPI = 2
- 15% branches, CPI = 1
Amdahl’s Law

Execution time after improvement =

\[
\frac{\text{execution time affected by improvement}}{\text{amount of improvement}} + \text{execution time unaffected}
\]

Or:

Speedup is limited by popularity of improved feature

Corollary:

Make the common case fast

Caveat:

Law of diminishing returns