Performance

Hakim Weatherspoon

CS 3410

Computer Science

Cornell University

The slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, and Sirer.

Goals for today

Performance

- What is performance?
- How to get it?

Performance

Complex question

- How fast is the processor?
- How fast your application runs?
- How quickly does it respond to you?
- How fast can you process a big batch of jobs?
- How much power does your machine use?

Measures of Performance

Clock speed

- 1 KHz, 10^3 Hz: cycle is 1 millisecond, ms, (10^{-6})
- 1 MHz, 10^6 Hz: cycle is 1 microsecond, us, (10^{-6})
- 1 Ghz, 10^9 Hz: cycle is 1 nanosecond, ns, (10^{-9})
- 1 Thz, 10^{12} Hz: cycle is 1 picosecond, ps, (10^{-12})

Instruction/application performance

- MIPs (Millions of instructions per second)
- FLOPs (Floating point instructions per second)
  - GPUs: GeForce GTX Titan (2,688 cores, 4.5 Tera flops, 7.1 billion transistors, 42 Gigapixel/sec fill rate, 288 GB/sec)
- Benchmarks (SPEC)

Measures of Performance

CPI: "Cycles per instruction"→Cycle/instruction for **on average**

- **IPC** = 1/CPI
  - Used more frequently than CPI
  - Favored because "bigger is better", but harder to compute with
- Different instructions have different cycle costs
  - E.g., "add" typically takes 1 cycle, "divide" takes >10 cycles
- Depends on relative instruction frequencies

CPI example

- Program has equal ratio: integer, memory, floating point
- Cycles per insn type: integer = 1, memory = 2, FP = 3
- What is the CPI? (33% * 1) + (33% * 2) + (33% * 3) = 2
- *Caveat:* calculation ignores many effects
  - Back-of-the-envelope arguments only

Measures of Performance

General public (mostly) ignores CPI

- Equates clock frequency with performance!

Which processor would you buy?

- Processor A: CPI = 2, clock = 5 GHz
- Processor B: CPI = 1, clock = 3 GHz
- Probably A, but B is faster (assuming same ISA/compiler)

Classic example

- 800 MHz PentiumIII faster than 1 GHz Pentium4!
- Example: Core i7 faster clock-per-clock than Core 2
- Same ISA and compiler!

Meta-point: danger of partial performance metrics!

Measures of Performance

Latency

- How long to finish my program
  - Response time, elapsed time, wall clock time
  - CPU time: user and system time

Throughput

- How much work finished per unit time

Ideal: Want high throughput, low latency

… also, low power, cheap ($$) etc.

How to make the computer faster?

Decrease latency

Critical Path

- Longest path determining the minimum time needed for an operation
- Determines minimum length of clock cycle
  i.e. determines maximum clock frequency

Optimize for latency on the critical path

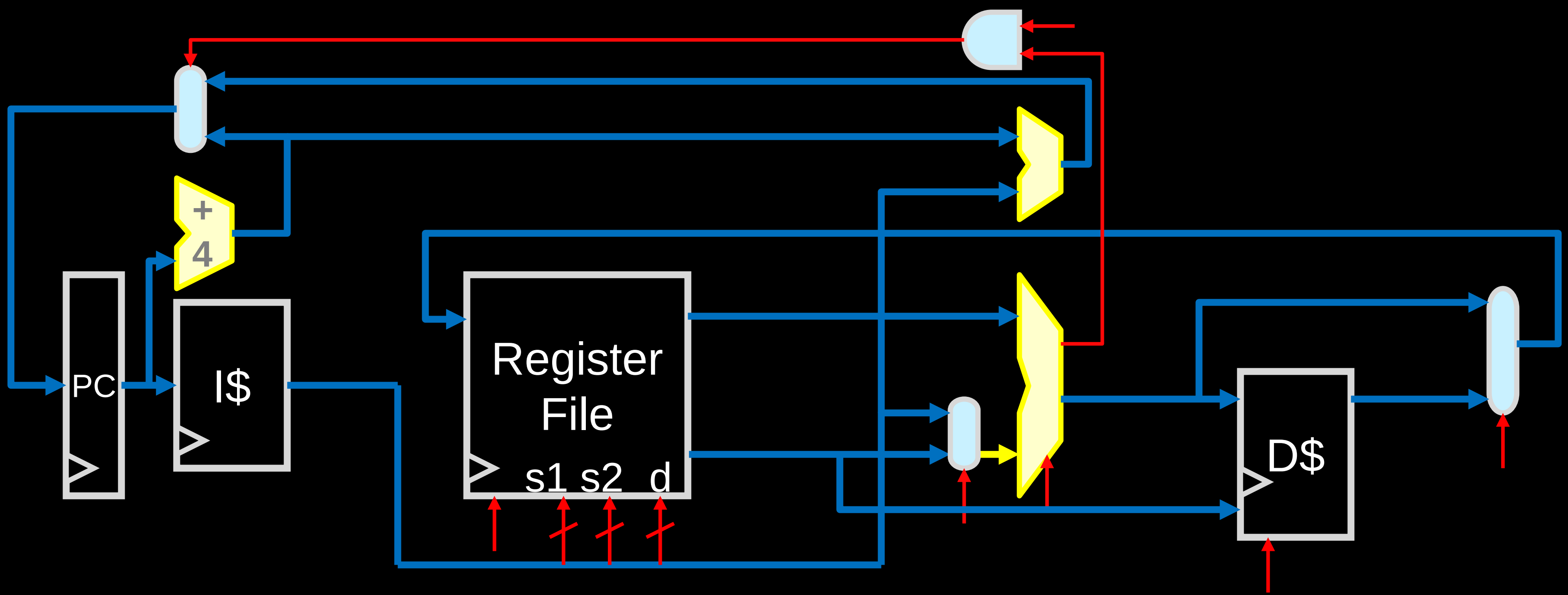
  - Parallelism (like carry look ahead adder)
  - Pipelining
  - Both

Latency: Optimize Delay on Critical Path

E.g. Adder performance

32 Bit Adder Design	Space	Time
Ripple Carry	≈ 300 gates	≈ 64 gate delays
2-Way Carry-Skip	≈ 360 gates	≈ 35 gate delays
3-Way Carry-Skip	≈ 500 gates	≈ 22 gate delays
4-Way Carry-Skip	≈ 600 gates	≈ 18 gate delays
2-Way Look-Ahead	≈ 550 gates	≈ 16 gate delays
Split Look-Ahead	≈ 800 gates	≈ 10 gate delays
Full Look-Ahead	≈ 1200 gates	≈ 5 gate delays

Review: Single-Cycle Datapath



Single-cycle datapath: true “atomic” F/EX loop

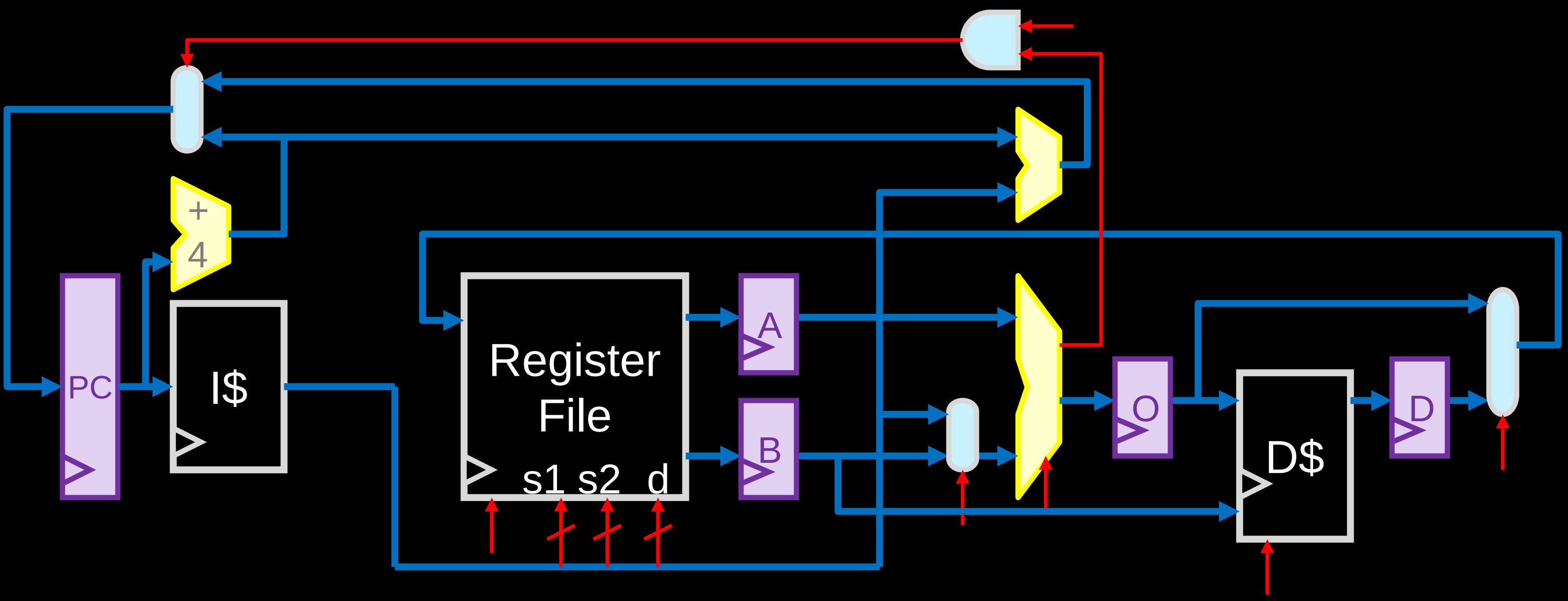
Fetch, decode, execute one instruction/cycle

+ Low CPI (later): 1 by definition

– Long clock period: accommodate slowest insn

(PC → I$ → RF → ALU → D$ → RF)

New: Multi-Cycle Datapath



Multi-cycle datapath: attacks slow clock

Fetch, decode, execute one insn over multiple cycles

Allows insns to take different number of cycles

±Opposite of single-cycle: short clock period, high CPI

Single- vs. Multi-cycle Performance

Single-cycle

- Clock period = 50ns, CPI = 1
- Performance = **50ns/insn**

Multi-cycle: opposite performance split

+ Shorter clock period
– Higher CPI

Example

- branch: 20% (**3** cycles), ld: 20% (**5** cycles), ALU: 60% (**4** cycle)
- Clock period = **11ns**, CPI = (20%*3)+(20%*5)+(60%*4) = 4
  – Why is clock period 11ns and not 10ns?
- Performance = **44ns/insn**

Aside: CISC makes perfect sense in multi-cycle datapath

Multi-Cycle Instructions

But what to do when operations take diff. times?

E.g: Assume:

- load/store: 100 ns ⟵ 10 MHz
- arithmetic: 50 ns ⟵ 20 MHz
- branches: 33 ns ⟵ 30 MHz

ms = 10^{-3} second
us = 10^{-6} seconds
ns = 10^{-9} seconds
ps = 10^{-12} seconds

Single-Cycle CPU

10 MHz (100 ns cycle) with

- 1 cycle per instruction

Multi-Cycle Instructions

Multiple cycles to complete a single instruction

E.g: Assume:

- load/store: 100 ns $\leftarrow$ 10 MHz
- arithmetic: 50 ns $\leftarrow$ 20 MHz
- branches: 33 ns $\leftarrow$ 30 MHz

ms = 10^{-3} second
us = 10^{-6} seconds
ns = 10^{-9} seconds
ps = 10^{-12} seconds

Which one is faster: Single- or Multi-Cycle CPU?

Single-Cycle CPU

10 MHz (100 ns cycle) with

– 1 cycle per instruction

Multi-Cycle CPU

30 MHz (33 ns cycle) with

- 3 cycles per load/store
- 2 cycles per arithmetic
- 1 cycle per branch

Cycles Per Instruction (CPI)

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:

Multi-Cycle @ 30 MHz

Single-Cycle @ 10 MHz

Total Time

CPU Time = # Instructions x CPI x Clock Cycle Time

sec/prgrm = Instr/prgm x cycles/instr x seconds/cycle

Instructions per program: "dynamic instruction count"

- Runtime count of instructions executed by the program
- Determined by program, compiler, ISA

Cycles per instruction: "CPI" (typical range: 2 to 0.5)

- How many *cycles* does an instruction take to execute?
- Determined by program, compiler, ISA, micro-architecture

Seconds per cycle: clock period, length of each cycle

- Inverse metric: cycles/second (Hertz) or cycles/ns (Ghz)
- Determined by micro-architecture, technology parameters

For lower latency (=better performance) minimize all three

- Difficult: ***often pull against one another***

Total Time

CPU Time = # Instructions x CPI x Clock Cycle Time

sec/prgrm = Instr/prgm x cycles/instr x seconds/cycle

E.g. Say for a program with 400k instructions, 30 MHz:

CPU [Execution] Time = ?

Example

Goal: Make Multi-Cycle @ 30 MHz CPU (15MIPS) run 2x faster by making arithmetic instructions faster

Instruction mix (for P):

- 25% load/store, CPI = 3
- 60% arithmetic, CPI = 2
- 15% branches, CPI = 1

Example

Goal: Make Multi-Cycle @ 30 MHz CPU (15MIPS) run 2x faster by making arithmetic instructions faster

Instruction mix (for P):

- 25% load/store, CPI = 3
- 60% arithmetic, CPI = 2
- 15% branches, CPI = 1

Goal: Make processor run 2x faster,
i.e. 30 MIPS instead of 15 MIPS

Amdahl's Law

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

- Don't optimize 1% to the detriment of other 99%
- Don't over-engineer capabilities that cannot be utilized

Caveat: Law of diminishing returns

Performance Recap