CSE 5600
Computer Systems Architecture

Performance

1

This Unit

"Speed is not everything but it's kilometers ahead of whatever is in second place."
—Ed McCreight, The Dragon Computer System
Xerox PARC September, 1984

- **Metrics**
  - Latency and throughput
- **Reporting performance**
  - Benchmarking and averaging
- **CPU performance equation**

2

Performance: Latency vs. Throughput

- **Latency (execution time)**: time to finish a fixed task
- **Throughput (bandwidth)**: number of tasks in fixed time
  - Different: exploit parallelism for throughput, not latency
  - Often contradictory (latency vs. throughput)
    - Will see many examples of this
  - Choose definition of performance that matches your goals
    - Single scientific program: latency; web server: throughput?

3

Problem #1: Car vs. Bus

Car: speed = 60 miles/hour, capacity = 5
Bus: speed = 20 miles/hour, capacity = 60

Task: transport passengers 10 miles

	Latency (min)	Throughput (PPH)
Car		
Bus		





4

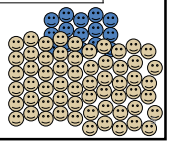
Problem #1: Car vs. Bus

Car: speed = 60 miles/hour, capacity = 5
Bus: speed = 20 miles/hour, capacity = 60

Task: transport passengers 10 miles

	Latency (min)	Throughput (PPH)
Car	10 min	15 PPH
Bus	30 min	60 PPH





5

Comparing Performance

- A is X times faster than B if

$$\text{Latency}(A) = \frac{\text{Latency}(B)}{X}$$

$$\text{Throughput}(A) = \text{Throughput}(B) \cdot X$$

- A is Y% faster than B if

$$\text{Latency}(A) = \frac{\text{Latency}(B)}{1+Y/100}$$

$$\text{Throughput}(A) = \text{Throughput}(B) \cdot (1+Y/100)$$

6

Problem #2: Car vs. Bus Revisited

- **Latency**
  Car = 10 min, Bus = 30 min
  - Car is __ times faster than bus
  - Car is _% faster than bus
- **Throughput**
  Car = 15 PPH, Bus = 60 PPH
  - Bus is _ times faster than car
  - Bus is _% faster than car



Problem #2: Car vs. Bus Revisited

- **Latency**
  Car = 10 min, Bus = 30 min
  - Car is 3 times faster than bus
  - Car is 200% faster than bus
- **Throughput**
  Car = 15 PPH, Bus = 60 PPH
  - Bus is 4 times faster than car
  - Bus is 300% faster than car

Reporting Performance

Benchmarking & Averaging

Processor Performance and Workloads

Q: what does Latency(ChipA) or Throughput(ChipA) mean?
A: nothing, there must be some associated workload

- **Workload**: set of tasks someone cares about
  → Latency(Task1, ChipA) (car/bus Task = drive ppl 10 miles)
- **Benchmarks**: standard workloads
  - Used to compare performance across machines
  - Are/highly representative of actual programs people run
- **Micro-benchmarks**: non-standard non-workloads
  - Tiny programs used to isolate certain aspects of performance
  - Not representative of complex behaviors of real applications
  - Frequently helpful to examine isolated performance questions

SPEC Benchmarks

- **SPEC (Standard Performance Evaluation Corporation)**
  - Consortium that collects, standardizes, and distributes benchmarks, http://www.spec.org/
  - Post **SPECmark** results for different processors
    - 1 number that represents performance for entire suite
  - Benchmark suites for CPU, Java, I/O, Web, Mail, etc.
  - Updated every few years: so companies don't target benchmarks
- **SPEC CPU 2006**
  - 12 "integer": bzip2, gcc, perl, hmmer (genomics), h264, …
  - 17 "floating point": wrf (weather), povray, sphynx3 (speech)…
  - Written in C/C++ and Fortran
- **SPEC CPU 2017**
  - 2 "integer" suites: latency vs. throughput
  - 2 "floating point" suites: latency vs. throughput

Other Benchmarks

- **Parallel benchmarks**
  - SPLASH2: Stanford Parallel Applications for Shared Memory
  - NAS: another parallel benchmark suite
  - SPECopenMP: parallelized versions of SPECfp
  - SPECjbb: Java multithreaded database-like workload
- **Transaction Processing Council (TPC)**
  - TPC-C: On-line transaction processing (OLTP)
  - TPC-H/R: Decision support systems (DSS)
  - TPC-W: E-commerce database backend workload
  - Have parallelism (intra-query and inter-query), heavy I/O, memory
- **Benchmarks for other domains**
  - DIBS: Data Integration Benchmark Suite (from our group at WashU)
  - MiBench: Embedded systems (from Michigan)
  - MediaBench: Media applications (out of UCLA)
- **Companies have internal benchmarks**
  - What's going to be important in the future?
  - Overfitting ☹

Mean (Average) Performance Numbers

3 Types of Means

- **Arithmetic**
  - for units that are proportional to time (*e.g.*, latency)
- **Harmonic**
  - for units that are inversely proportional to time (*e.g.*, throughput)
- **Geometic**
  - For unitless quantities (*e.g.*, speedup ratios)

Know when to use which one & how it is computed.

13

13

Arithmetic Mean

For units that are proportional to time (*e.g.*, latency)

Chip A, N programs:

$$\frac{\Sigma_{i=1..N}\ \text{Latency}(P_i, A)}{N}$$

You can add latencies, but not throughputs

- Latency(P1+P2, A) = Latency(P1, A) + Latency(P2, A)
- Throughput(P1+P2, A) ≠ Throughput(P1, A) + Throughput(P2, A)
  - 1 mile @ 10 miles/hour + 1 mile @ 100 miles/hour
  - Average is **not** 55 miles/hour
  - Need a different mean….

14

14

Harmonic Mean

For units that are inversely proportional to time (*e.g.*, throughput)

Chip A, N programs:

$$\frac{N}{\Sigma_{i=1..N}\ 1/\text{Throughput}(P_i, A)}$$

P_1: 1 mile @ 30 miles/hour
P_2: 1 mile @ 90 miles/hour

$$\frac{2}{1/30 + 1/90} = 45 \text{ mph}$$

15

15

Geometric Mean

For unitless quantities (*e.g.*, speedup ratios)

$$\sqrt[N]{\Pi_{i=1..N}\ \text{Speedup}(P_i, A)}$$

16

16

Performance Equation(s)

17

17

Processor Performance Equation

Program runtime:

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

(Execution time) (Dynamic instruction count) (CPI) (Clock period)

- **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)
  - About 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***

18

18

Cycles per Instruction (CPI)

- **CPI**: Cycle/instruction for a program **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 ops, floating point
  - Cycles per instruction: integer = 1, memory = 2, FP = 3
  - What is the CPI? (33% × 1) + (33% × 2) + (33% × 3) = 2
  - **Caveat**: this sort of calculation ignores many effects
    - Back-of-the-envelope arguments only

Problem #3: CPI Example

- Assume a processor with instruction frequencies and costs
  - Integer ALU: 50%, 1 cycle
  - Load: 20%, 5 cycle
  - Store: 10%, 1 cycle
  - Branch: 20%, 2 cycle
- Which change would improve performance more?
  A: "Branch prediction" to reduce branch cost to 1 cycle?
  B: "Cache" to reduce load cost to 3 cycles?
- Compute CPI

	INT	LD	ST	BR	CPI
Base					
A					
B					



Problem #3: CPI Example

- Assume a processor with instruction frequencies and costs
  - Integer ALU: 50%, 1 cycle
  - Load: 20%, 5 cycle
  - Store: 10%, 1 cycle
  - Branch: 20%, 2 cycle
- Which change would improve performance more?
  A: "Branch prediction" to reduce branch cost to 1 cycle?
  B: "Cache" to reduce load cost to 3 cycles?
- Compute CPI

	INT	LD	ST	BR	CPI
Base	0.5 x 1	0.2 x 5	0.1 x 1	0.2 x 2	**2.0**
A	0.5 x 1	0.2 x 5	0.1 x 1	**0.2 x 1**	**1.8**
B	0.5 x 1	**0.2 x 3**	0.1 x 1	0.2 x 2	**1.6**

(winner)

MHz (MegaHertz) and GHz (GigaHertz)

- 1 Hertz = 1 cycle per second
  1 GHz is 1 cycle per nanosecond, 1 GHz = 1000 MHz
- 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!
  - Recent example: Core i7 faster clock-per-clock than Core 2
  - Same ISA and compiler!
- **Meta-point: danger of partial performance metrics!**

MIPS (performance metric, not the ISA)

- (Micro) architects often ignore dynamic instruction count
  - Typically have one ISA, one compiler → treat it as fixed
- CPU performance equation becomes

$$\text{Latency: } \frac{\text{seconds}}{\text{insn}} = \frac{\text{cycles}}{\text{insn}} \times \frac{\text{seconds}}{\text{cycle}}$$

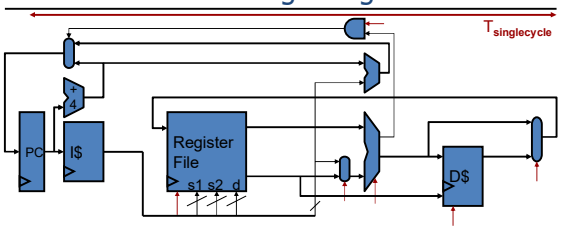
$$\text{Throughput: } \frac{\text{insns}}{\text{second}} = \frac{\text{insns}}{\text{cycle}} \times \frac{\text{cycles}}{\text{second}}$$

- **MIPS** (millions of instructions per second)
  - **Cycles / second**: clock frequency (in MHz)
  - Ex: CPI = 2, clock = 500 MHz → 0.5 x 500 MHz = 250 MIPS
- Pitfall: may vary inversely with actual performance
  - Compiler removes insns, program faster, but lower MIPS
  - Work per instruction varies (multiply vs. add, FP vs. integer)

Latency vs. Throughput Revisited

- Two views of performance: latency vs. throughput
- Two scopes of performance: instruction vs. program
- **Single instruction latency**
  - Doesn't matter: programs comprised of billions+ of insns
  - Difficult to reduce anyway
  - Making 1 insn faster doesn't help unless it's **the** slowest insn
- **Instruction throughput** → *program latency or throughput*
  + Can improve using ***parallelism***
    - Multiple cores (more units executing instructions)… more later
    - Inter-instruction parallelism example: pipelining

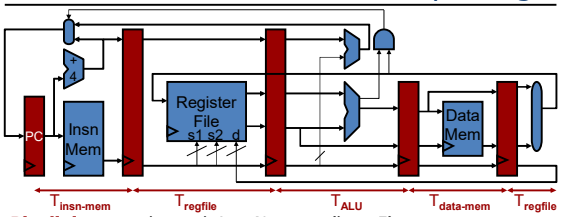
In the Beginning...



...was the single-cycle data path

- Fetch, decode, execute one complete insn every cycle
- \+ Low CPI: 1 by definition
- – Long clock period: set by longest insn (logic + wire delays)

Inter-Instruction Parallelism: Pipelining



Pipelining: cut datapath into N stages (here 5)

- Separate each stage of logic by latches
- Clock period: maximum logic + wire delay of any stage
  = max($T_{insn-mem}$, $T_{regfile}$, T_{ALU}, $T_{data-mem}$)
- Base CPI = 1, but actual CPI > 1: pipeline stalls a lot
- Individual insn latency increases (pipeline overhead), that's ok

Pipelining: Clock Frequency vs. IPC

- Increase number of pipeline stages ("pipeline depth")
  - Keep cutting datapath into finer pieces
  - \+ Increases clock frequency (decreases clock period)
    - **Latch overhead & unbalanced stages** cause sub-linear scaling
    - Double the number of stages won't quite double the frequency
  - – Decreases IPC (increase CPI)
    - More pipeline "hazards", higher branch penalty
    - Memory latency relatively higher (same absolute lat., more cycles)
  - – Result: at some point, deeper pipelines decrease performance
  - "Optimal" pipeline depth is program and technology specific
- Classic example
  - Pentium III: 12 stage pipeline, 800 MHz
  - Pentium 4: 22 stage pipeline, 1 GHz (and *slower* due to IPC)

Note: clock frequency implies CPU clock. Other system components have own clocks (or not).

Problem #4: CPI and Clock Frequency

1 GHz processor with

- 80% non-memory instructions @ 1 cycle
- 20% memory insns @ 6 nanoseconds (= 6 cycles)

Double the core clock frequency?

- Increasing processor clock doesn't accelerate memory!
  - Non-memory instructions retain 1-cycle latency
  - Memory instructions now have 12-cycle latency

Infinite clock frequency?

- *Hello, Amdahl's Law!*

	Non-Mem	Mem	CPI	MIPS	Speedup
1 GHz					
2 GHz					
∞ GHz					

Problem #4: CPI and Clock Frequency

1 GHz processor with

- 80% non-memory instructions @ 1 cycle
- 20% memory insns @ 6 nanoseconds (= 6 cycles)

Double the core clock frequency?

- Increasing processor clock doesn't accelerate memory!
  - Non-memory instructions retain 1-cycle latency
  - Memory instructions now have 12-cycle latency

Infinite clock frequency?

- *Hello, Amdahl's Law!*

	Non-Mem	Mem	CPI	MIPS	Speedup
1 GHz	0.8 x 1	0.2 x 6	2.0	500	
2 GHz	0.8 x 1	0.2 x 12	3.2	625	1.25 (<< 2)
∞ GHz	1M insn / (200K x 6 ns)			833	1.66

Measuring CPI

- How are CPI and execution-time actually measured?
  - Execution time? stopwatch timer (Unix "time" command)
  - CPI = execution time / (clock period x dynamic insn count)
  - How is dynamic instruction count measured?
- More useful is CPI breakdown (CPI_{CPU}, CPI_{MEM}, etc.)
  - So we know what performance problems are and what to fix
  - Hardware event counters
    - Available in most processors today
    - One way to measure dynamic instruction count
    - Calculate CPI using counter frequencies / known event costs
  - Cycle-level micro-architecture simulation (*e.g.*, SimpleScalar)
    - \+ Measure exactly what you want … and impact of potential fixes!
    - Method of choice for many micro-architects
  - Hardware emulation (e.g., on FPGAs) becoming common

Performance Rules of Thumb

Amdahl's Law: *"Make the common case fast"*

- Literally: total speedup limited by non-accelerated piece
- Example: can optimize 50% of program A
  - Even "magic" optimization that makes this 50% disappear…
  - …only yields a 2X speedup

Corollary: **build a balanced system**

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

Design for actual performance, **not peak performance**

- Peak perf: "Performance you are guaranteed not to exceed"
- Greater than "actual" or "average" or "sustained" performance
  - Why? Caches misses, branch mispredictions, limited ILP, *etc.*
- For actual performance X, machine capability must be > X

31

31

Summary

- Latency:

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

- **Instructions / program**: dynamic instruction count
  - Function of program, compiler, instruction set architecture
- **Cycles / instruction**: CPI
  - Function of program, compiler, ISA, micro-architecture
- **Seconds / cycle**: clock period
  - Function of micro-architecture, technology parameters

- Optimize each component
  - CSE 5600 focuses mostly on CPI (caches, parallelism)
  - …but some on dynamic instruction count (compiler, ISA)
  - …and some on clock frequency (pipelining, technology)

32

32