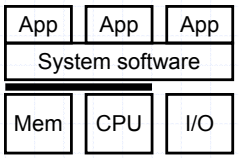
CIS 371
Computer Organization and Design

Unit 3: Performance

This Unit

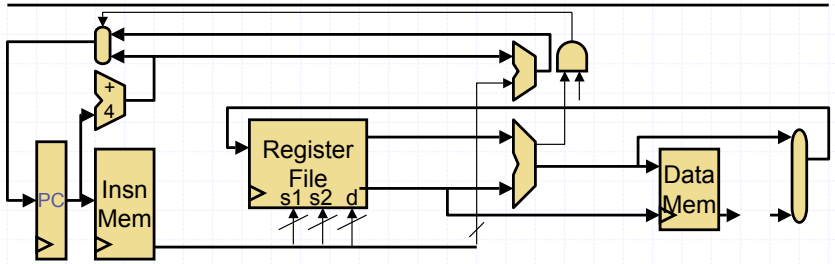


- Performance
  - Latency and throughput
  - Benchmarking
  - CPU performance equation

Readings

- P+H
  - Chapter 4

240 → 371



- CSE 240: build something that works
- CSE 371: build something that works "well"
  - "well" means "high-performance" but also cheap, low-power, etc.
  - Mostly "high-performance"
  - So, what is the performance of this?
  - What is performance?

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 (e.g., bread)
  - Often contradictory (latency **vs.** throughput)
    - Will see many examples of this
  - Choose definition of performance that matches your goals
    - Scientific program? Latency, web server: throughput?

- Example: move people 10 miles
  - Car: capacity = 5, speed = 60 miles/hour
  - Bus: capacity = 60, speed = 20 miles/hour
  - Latency: **car = 10 min**, bus = 30 min
  - Throughput: car = 15 PPH (count return trip), **bus = 60 PPH**

Comparing Performance

- A is X times faster than B if
  - Latency(A) = Latency(B) / X
  - Throughput(A) = Throughput(B) * X
- A is X% faster than B if
  - Latency(A) = Latency(B) / (1+X/100)
  - Throughput(A) = Throughput(B) * (1+X/100)

- Car/bus example
  - Latency? Car is 3 times (and 200%) faster than bus
  - Throughput? Bus is 4 times (and 300%) faster than car

Processor Performance and Workloads

- Q: what does latency(Pentium) or thruput(Pentium) mean?
- A: nothing, there must be some associated workload
  - **Workload**: set of tasks someone (you) cares about

- **Benchmarks**: standard workloads
  - Used to compare performance across machines
  - Either are or 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
  - Examples: towers-of-hanoi, 8-queens, etc.

SPEC Benchmarks

- SPEC: Standard Performance Evaluation Corporation
  - http://www.spec.org/
  - Consortium that collects, standardizes, and distributes benchmarks
  - Suites for CPU, Java, I/O, Web, Mail, OpenMP (multithreaded), etc.
  - Updated every few years: so companies don't target benchmarks
  - Post **SPECmark** results for different processors
    - 1 number that represents performance for entire suite
  - CPU 2006: 29 CPU-intensive C/C++/Fortran programs
    - "integer": bzip2, gcc, perl, hmmer (genomics), h264, etc.
    - "floating-point": wrf (weather), povray, sphynx3 (speech), etc.

- TPC: Transaction Processing Council
  - Like SPEC, but for web/database server workloads
  - Much heavier on memory, I/O, network, than on CPU
  - Doesn't give you the source code, only a 'description'

SPECmark

- Reference machine: Sun SPARC 10
- Latency SPECmark
  - For each benchmark
    - Take odd number of samples: on both machines
    - Choose median
    - Take latency ratio (Sun SPARC 10 / your machine)
  - Take GMEAN of ratios over all benchmarks
- Throughput SPECmark
  - Run multiple benchmarks in parallel on multiple-processor system
- Recent SPECmark latency leaders
  - SPECint: Intel 2.3 GHz Core2 Extreme (3119)
  - SPECfp: IBM 2.1 GHz Power5+ (4051)

Mean (Average) Performance Numbers

- **Arithmetic**: (1/N) * $\sum_{P=1..N}$ Latency(P)
  - For units that are proportional to time (e.g., latency)
- 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 @ 30 miles/hour + 1 mile @ 90 miles/hour
    - Average is **not** 60 miles/hour
- **Harmonic**: N / $\sum_{P=1..N}$ 1/Throughput(P)
  - For units that are inversely proportional to time (e.g., throughput)
- **Geometric**: $\sqrt[N]{\prod_{P=1..N}}$ Speedup(P)
  - For unitless quantities (e.g., speedups)

CPU Performance Equation

- Multiple aspects to performance: helps to isolate them
- Latency = seconds / program =
  - (insns / program) * (cycles / insn) * (seconds / cycle)
  - **Insns / program**: dynamic insn count = f(program, compiler, ISA)
  - **Cycles / insn**: CPI = f(program, compiler, ISA, micro-arch)
  - **Seconds / cycle**: clock period = f(micro-arch, technology)
- For low latency (better performance) minimize all three
  - Difficult: often pull against one another
  - Example we have seen: RISC vs. CISC ISAs
    - ± RISC: low CPI/clock period, high insn count
    - ± CISC: low insn count, high CPI/clock period

MIPS (performance metric, not the ISA)

- Factor out dynamic insn count, CPU equation becomes…
  - Latency: seconds / insn = (cycles / insn) * (seconds / cycle)
  - Throughput: **insns / second** = (insns / cycle) * (cycles / second)
- **MIPS** (millions of insns per second): insns / second * 10^{-6}
  - **Cycles / second**: clock frequency (in MHz)
  - Example: CPI = 2, clock = 500 MHz, what is MIPS?
    - 0.5 * 500 MHz * 10^{-6} = 250 MIPS
- MIPS is OK for micro-architects
  - Typically work in one ISA/one compiler, treat insn count as fixed
- Not OK for general public
  - Processors with different ISAs/compilers have incomparable MIPS
  - Wait, it gets worse…

Mhz (MegaHertz) and Ghz (GigaHertz)

- 1 Hertz = 1 cycle per second
  1 Ghz is 1 cycle per nanosecond, 1 Ghz = 1000 Mhz
- Micro-architects often ignore instruction count…
- … but general public (mostly) also 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!
  - Same ISA and compiler!
- **Meta-point: danger of partial performance metrics!**

Non-CPU Performance Equation

- Clock frequency implies CPU clock
  - Other system components have their own clocks (or not)
  - E.g., increasing processor clock doesn't accelerate memory
- Example
  - Processor A: CPI_{CPU} = 1, CPI_{MEM} = 1, clock = 500 MHz
  - What is the speedup if we double clock frequency?
  - Base: CPI = 2 → IPC = 0.5 → MIPS = 250
  - New: CPI = **3** → IPC = 0.33 → MIPS = 333
    - Clock *= 2 → CPI_{MEM} *= 2
  - Speedup = 333/250 = 1.33 << 2
- What about an infinite clock frequency?
  - Only a 2X (factor of 2) speedup
  - Example of Amdahl's Law

Amdahl's Law

- 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
- For consumers: buy a balanced system
- For microarchitects: build a balanced system
  - **MCCF (Make Common Case Fast)**
  - Focus your efforts on things that matter

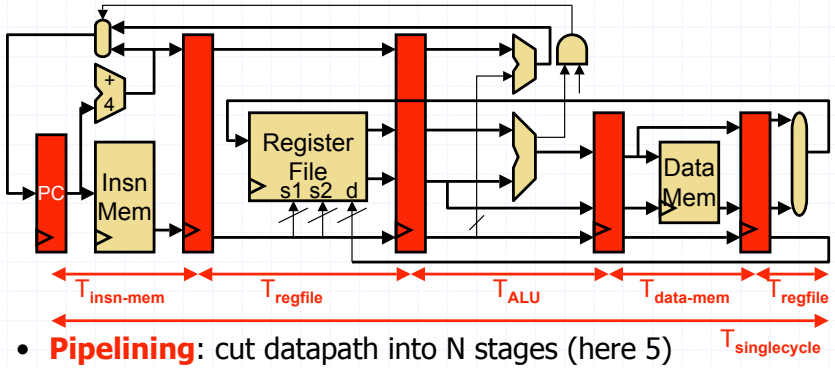
How Can We Make Common Case Fast?

- If we don't know what CC is?
- How is CPI actually measured?
  - Execution time: time (Unix): wall clock / CPU + system
  - CPI = CPU time / (clock frequency * dynamic insn count)
- How is dynamic insn count measured?
  - Hardware event counters: e.g., LC3/P37X insn counter
- More useful is CPI breakdown (CPI_{CPU}, CPI_{MEM}, etc.)
  - So we know what performance problems are and what to fix
  - Hardware event counters: e.g., LC3/P37X branch/load stall counters
    + Accurate
    – Can't measure everything or evaluate modifications
  - Cycle-level micro-architecture simulation: e.g., SimpleScalar
    + Measure exactly what you want, evaluate potential fixes
    – Burden of accuracy is on the simulator writer

Latency vs. Throughput Revisited

- Latency and throughput: two views of performance …
  - … at the program level
  - … not at the insn level
- Single insn latency
  - Nobody cares: programs comprised of [billions]+ of insns
  - Difficult to reduce anyway
- As number of dynamic instructions is large…
  - Insn throughput → program latency or throughput
  - \+ Can reduce using inter-insn ***parallelism***
    - Most important example: pipelining

Inter-Insn Parallelism: Pipelining

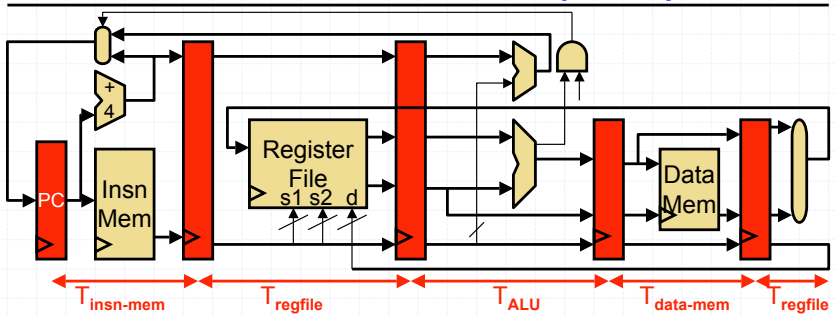


- **Pipelining**: cut datapath into N stages (here 5)
  - One insn in each stage in each cycle
  - \+ Clock period = MAX($T_{insn-mem}$, $T_{regfile}$, T_{ALU}, $T_{data-mem}$)
  - \+ Base CPI = 1: insn enters and leaves every cycle
  - – Actual CPI > 1: pipeline must often stall
  - Individual insn latency increases (pipeline overhead), not the point

Pipelining: Clock Frequency vs. IPC

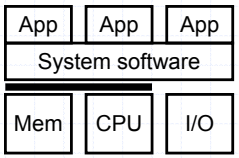
- Increase number of pipeline stages
  - \+ Increases clock frequency (decreases clock period)
  - – Decreases IPC (increase CPI)
  - – At some point, actually causes performance to decrease
  - "Optimal" pipeline depth is program and technology specific
- Remember example
  - PentiumIII: 12 stage pipeline, 800 MHz
  - Pentium4: 22 stage pipeline, 1 GHz
    - – Actually slower (because of lower IPC)
  - Core2: 15 stage pipeline
    - \+ Intel learned its lesson

What Determines Clock Frequency?



- Technology (transistors & wires), micro-architecture (ALUs, pipeline)
- Clock period = MAX($T_{insn-mem}$, $T_{regfile}$, T_{ALU}, $T_{data-mem}$)
  - But which one of these is largest?
  - Simple model: each gate has delay of '1', wires have no delay
    - T_{ALU} is largest: N-bit ripple-carry has 2N gate delays
  - Reality: ALUs don't use ripple-carry, wire delay dominates storage

Summary



- Performance
  - Latency and throughput, metrics
  - CPU performance equation
- Next
  - Fast integer arithmetic