[redacted]

CSE141 Midterm

Performance

Metrics:

① Latency: $t_{stop} - t_{start}$ (seconds). A task on a CPU has latency.
- speed/performance $= \frac{1}{latency}$, speedup $= \frac{latency_{before}}{latency_{after}}$ { >1: faster; <1: slower.
- importance: waiting / human vision / due soon

② Bandwidth: work per time; bytes/second, instruction/seconds, etc.
- latency $= \frac{work}{bandwidth}$. e.g. write 10MB at 10MB/s : 1s

Key Equations

① Amdahl's Law: optimizations × (generally) uniformly affect the entire program
- $Speedup_{total} = \frac{1}{x_1/s_1 + x_2/s_2 + \cdots 1 - x_1 - x_2 \cdots - x_n}$ x_i: fraction of applications, s_i: speedup.
- corollaries: common —optimize→ uncommon —final→ new common ⇒ { global optimize; divide up dif
  * make common case ... fast until there is no common

② CPU Performance Equation
- Execution time = IC × CPI × CT $= \frac{cycles}{instrn}$ (CPI) × instructions (IC) × $\frac{seconds}{cycles}$ (CT)
- CPI: average cycles, CT $= \frac{1}{frequency}$ (G: 10^9, M: 10^6), IC: dynamic instructions

A) Reduce Instruction Count: Algorithm/Compiler
- { O0: no optimization, variable on stacks. lots of loads/stores; O1: × loads/stores, fewer movs, nothing on stack } { common sub elimination; constant propagation; function in-line
- inline → constant → loop unroll → partial evaluation (substitute computation)
- Impacts: program input, instruction set

B) Improve CPI: $\frac{total\ cycles}{total\ instructions}$ = average latency of an instruction
- { Program: float/memory, larger CPI; Inputs: exercise different parts of the program; Compile: which instruction. e.g. O0 vs O1

C) Affect CT: processor design (Cycle time_{min} = delay max along critical path)
Manufacturing variation (transistor chars). software policy

③ Power and Energy Consumption

A) Power $= \frac{Energy}{Time} = J/s =$ Watt; $P = V^2FaC + P_{idle}$ (voltage, activity factor, frequency, capacitance (area), idle power consumption)
- P increases with area (C) and switching factor (a)
- Pidle proportional to the area of the chip
- V and F are linearly related. $P \sim F^3aC + P_{idle}$: F↑, Power↑

B) Energy = Power × Time = $(V^2FaC + P_{idle}) \times (CPI \times IC \times CT)$
- Clock freq ↑ : E↑, P↑, ET↓

Benchmark well-defined computation used to quantify system characteristics
- types { Micro: measure specific aspect; Macro: stand app using standard inputs · characteristics { Realistic, portable, easy to use; own datasets, useful meas; capture compiler/architecture; PoC: run a user-specific program
- Spec2017: industry consortium (vetos), open-source, all in C/C++/Perl

x86 Assembly

ISA
Hardware – ISA { instruction format; instruction operation → software; interaction with memory

16	32	64	
ax	eax	rax	accumulator
bx	ebx	rbx	base
cx	ecx	rcx	counter

b	s	w	l	q	
8	16	16	32	64	bits

x86
- instruction src1 src2 dst;
- %<reg>: register, \$imm: immediate, \$label: label. m(%eax,%ebx,n) = Mem[m+(R[eax]+R[ebx]×n)]
- %eax: reg value; (%eax): Mem(R[eax]); n(%eax): Mem(R+n);
- Compare: cmpl %eax %ebx : %eax − %ebx; e.g. cmp %eax, %ebx / jge label ⇒ %ebx >= %eax
  - ZF: = 1 if s1 and s2 are equal; SF: 1 if s2 < s1.
- Function call: ret: return address; leave: restore call stack pointer

Types { CISC: complex to compiler, human readable, consume to add features [x86, VAX] (interface ✓, implementation ×) variable { inst, length; RISC: compiler optimizable [Alpha, MIPS, SPARC]. fixed: inst. length, behavior (interface ×, implementation ✓)
- x86 → RISC-like instructions → execution

Efficient Execution

Instruction Execution: fetch → Decode → RF read → { Arithm, mem, branch } → next PC { single inst → 6 cycle; CPI↑, energy↑

Pipeline use all stages at a time (cycles for an instrn remain constant)
- limitation: IC [ISA, compiler], CPI = 1 [Best case], CT [critical logic path in slowest stage]

① Control hazard: fetch stage × know next
- Problem: Perform ~ CPI = 6 by stalling, Energy ~ a is small $\frac{CPI_a \times IC_a + CPI_b \times IC_b}{IC_a + IC_b}$
- ⇒ Speculative execution: predict on prior, detect misprediction
  E.g. 1-bit local predictor $T[A\%2^N] == 1$ predict T else NT; $T[A\%2^N]$ = taken ? 1 : 0
- Misspeculation: CPI↑, pipeline flushing
  - IC determined by non-speculative, compiler/ISA; CPI: cycles wasted, Pidle
  - Energy: a = 1 since all are busy. CPI↑ wasted energy; Non-speculative worse

② Data hazard: Forward, connections bypass the reg file

③ x86 terrible: micro-op. decouple instructions for flexibility. average 1.3 uops/x86
Decode decompose instruction into uops → decode queue

More performance

① Deeper pipeline: Fetch [|] → Decode [|], then CT↓ 50%, CPI↑ { cost of misprediction; Data × resolve by forwarding (more than double)
× perfect: pipeline adds overlay, ↑ space & energy
- Impact. ET↓ (CT↓50%), Energy (aC↑ extra area, switching). Power (F doubles)

② Wider pipeline: two wide (rstr. data dependence)
- Impact: ET↓ (CPI↓50%) some, Energy (aC↑ bigger reg file)↑ a bit
- wider: more hazards, idle units, overhead: long-latency complications

③ Out-of-Order Execution:

A) Data Dependency – instruction reads correct values / order
- { RAW; WAW; WAR } false dependency: no data flows but A must execute before B
- critical path: longest sequence of RAW, WAW, and/or WAR-dependent instr
- formulas: $CPI = \frac{CP}{\#\ instructions}$, average $ILP = \frac{1}{CPI}$, $IC = ILP \times CP = \frac{1}{CPI} \times CP$
  $ET = IC \times CPI \times CT = IC \times \frac{CP}{IC} \times CT = CP \times CT$
- effective latency: its contribution to CP (only RAW constrains parallelism)

B) Register renaming: RAT ~ Architecture → Physical registers. inst: log(pr) × ar
- ILP: examine larger region of the instruction stream

C) Out of order issue [Tomasulo's Algo] (a = , = a, a = ; w, a.p1, a.p2, a.p3)
- Inputs { physical reg; broadcast from ALU after the instrn finishes → scheduler ⇒ ALU
- [past | retirement | ROB | fetch | future] speculation frontier; all prior branches resolved; reorder buffer
- Scheduler: issue instructions to ports [pipelined]. { reg value available; a free, compatible port; prioritize: old instrs

Modern Processor
- F →(x86) D →(uops) Decode Queue →(uops) ROB →(uops) rename RAT → scheduler → ALU (Backend); front; Speculation / Retirement control; Accuracy
- smaller uops retired each cycle → branch misprediction · = [acc] # branches
- Branch prediction ↑, ROB utilization ↑

Cache

formulas { # block offset bits = $\log_2$(block-size) bytes; # entries $= \frac{cache\ size\ (KB, 1024\ bytes)}{block\ sizes \times n}$; index = $\log_2$(# entries)
tag = address − # index − # offset
* block size: data transferred on miss
| tag | index | offset |, block size ↑ ⇒ miss ↓, # blocks ↓

localities: temporal ~ near in time, access same very soon; spatial ~ near in space, next access is close to last access.

cache misses:
- { compulsory: first access to the data; capacity: address requested AND fully-associative cache of same size exp a miss (cache isn't big enough); conflict: indexing caused collisions. addr AND full-ass cache of ... exp a hit

Set Associative index → set [block1 / block2]

store policies
- miss: write allocate (bring cache into cache), write no-allocate (notify lower cache) (notify lower cache the change when evicted)
- hit: write through (tell lower level cache what was changed), write back: mark as dirty

Prefetching
H: Stream buffer { stride: difference; confidence · impact: { idle bandwidth delay other request; energy: → lower ↑ energy
S: PREFETCH: insert your own preferences.