Performance and Power

Mainak Chaudhuri
Indian Institute of Technology Kanpur

Sketch

- Abstract model of computer
- Computer performance
- Amdahl's law
- Benchmarks
- Computer power
- Moore's law

Abstract model of computer

- Computer has an ISA
- The implementation of the ISA is an abstract five-state synchronous FSM
  - Each state change happens on posedge clock
  - State 0: fetch the instruction pointed to by program counter from memory; update program counter to point to the next instruction
  - State 1: decode the instruction to extract various fields and read source register operands
  - State 2: execute the instruction in ALU; compute address of load/store instructions; update program counter if control transfer instruction
  - State 3: access memory if load/store instruction
  - State 4: write result to destination register if the instruction produces a result

Abstract model of computer

- Multi-cycle implementation of ISA
  - Previous slide shows a five-cycle implementation
  - Five states represent five stages of instruction processing
  - Clock cycle time = longest stage latency
- Equivalently, one could implement all five stages in a large single combinational logic
  - Five stages would take one large single clock cycle
- Two designs are almost equivalent
  - We will further refine this statement later
  - For now, you may assume any of the two implementations

Abstract model of computer

- In both models, complexity of the ISA can influence the clock cycle time
  - If an instruction does a lot of things, that will lengthen the clock cycle time or may require more than five stages because execution of an instruction cannot be done in a single stage
    - Increases average number of cycles per instruction if we keep the cycle time constant in a multi-cycle design
  - Simple instructions can make clock cycle time smaller leading to high-frequency designs
- Which one is better?
  - Simple ISA with fast clock
    - Simple ISA leads to more instructions in a program
  - Complex ISA with slow clock
    - Complex ISA leads to less instructions in a program
  - Answer is not obvious

Why measure performance?

- Primarily two reasons
  - To measure how good a computer is
  - To understand why a computer is or is not performing as expected
- Performance of a computer is a function of the performance of individual programs that run on the computer
- A program's performance is best measured as the reciprocal of its execution time
- Equivalently, two computers' performance on the same program can be compared by looking at the reciprocal of respective execution times

CPI equation

- How do we measure execution time? Which are the determinant factors?
- Assume that we want to calculate the execution time of a program
  - Execution time = Clock cycles to execute the progam × clock cycle time
  - Executed clock cycles = number of executed instructions × average cycles per instruction
  - Execution time = instruction count × CPI × cycle time
  - Cycle time is also same as reciprocal of frequency (in appropriate unit)
  - Execution time equally depends on three components

CPI equation

- Each component can be improved to get reduced execution time
  - Reducing instruction count of a program normally depends on the instruction set of the processor and the smartness of the compiler e.g. separate equality check and branch instructions can be fused into one instruction such as bne or beq; similarly compiler can identify simple optimizations: ANDing with a mask and check instead of shift and AND and check
  - Reducing CPI depends on the implementation of the ISA (known as microarchitecture)
  - Frequency of a processor depends on semiconductor technology as well as the microarchitecture (not independent of CPI)

CPI equation

- Example
  - Consider a processor that does not have a FPSQRT instruction and FPSQRT is emulated in software (large CPI)
  - Frequency of FP instructions is 25%
  - Average CPI of these instructions is 4.0
  - Average CPI of non-FP instructions is 1.33
  - Frequency of FPSQRT operation is 2%
  - CPI of FPSQRT operation is 20.0
  - One design alternative is to reduce CPI of FPSQRT to 2
  - Other alternative is to reduce CPI of all FP operations to 2.0
  - Which one is better?

CPI equation

- Solution
  - Assume number of instructions = N, cycle time = t (in some unit)
  - $Time_{base}$ = (0.25 x 4 + 0.75 x 1.33)Nt = 2Nt
  - $Time_{design1}$ = $Time_{base}$ – 0.02N x 20t + 0.02N x 2t = 1.64Nt
  - $Time_{design2}$ = (0.25 x 2 + 0.75 x 1.33)Nt = 1.5Nt
  - Second design option is better

CPI equation

- Example
  - For a computer, CPI of arithmetic instructions is 1, CPI of load/store instructions is 10, CPI of branch instructions is 3
  - A program has 500 arithmetic instructions, 300 load/store instructions, 100 branch instructions
  - Design optimization: ISA is expanded with new more powerful arithmetic instructions so that the number of arithmetic instructions in the program reduces by 25% and clock cycle time increases by 10%; is this design better?
  - Without expanding the ISA, what if the arithmetic instruction CPI is improved? What is the maximum performance improvement?

CPI equation

- How to get these three parameters?
  - CPU designers normally use simulators to get exact behavior of program execution
    - Simulator is a piece of software that mimics the behavior of a computer
    - SPIM is a simple MIPS simulator
    - Simulators can be used to collect instruction count of a program and CPI
  - A user can exploit the performance counters of a computer to get a rough estimate of time spent on certain code segments and the number of instructions in those segments
    - Since frequency of a processor is known, these can be used to calculate CPI
  - Static profiling of the assembly language program can provide some information about instruction distribution

Amdahl's law

- Make the common case fast
  - No point investing time and money to optimize part of a system that gets invoked few times
  - Mathematical formulation:
    - Suppose a program takes time *t* to execute on a processor
    - A particular section of the program can be enhanced by some optimization in the processor; suppose *x* fraction of entire execution time is spent in this particular section of the program
    - The optimization in the processor can speed up execution of this section by *y* times
    - Therefore, overall speedup due to this optimization is

$$t / t_{new} = t / (t - tx + tx/y) = 1 / (1 - x + x / y)$$

    - Amdahl's law says that as y approaches infinity, the speedup saturates at 1/(1 – x) i.e., effort should be invested in this optimization only if x is large

Amdahl's law

- Look for portions of program that takes large amount of time to execute
  - Allocate resources and design time proportionate to execution time
  - As *x* increases the achieved speedup goes up for a fixed *y*; as *y* increases, speedup remains limited by *x*
  - Amdahl's law is usually used to compare design alternatives i.e. which design would bring more performance
  - Example: FP square root is critical in graphics applications; two design choices: implement a FP square root hardware to improve *sqrt* execution by 10 times or improve all FP instructions by 2 times; suppose FP sqrt takes 20% of execution time while 50% time is spent in all FP instructions in the current processor; which design choice is better?

Amdahl's law

- Amdahl's law can be used to derive upper bound on achievable speedup in a parallel computer
  - Suppose a sequential program takes time t to run on a single processor
  - A fraction s of this time is spent in executing inherently sequential portions of the program
  - The remaining time can be perfectly parallelized on arbitrary number of processors
  - Maximum achievable speedup = t / (s*t + (1 – s)*t / P) which is 1 / (s + (1 – s) / P) on P processors
  - In the limit, speedup gets capped at 1 / s
    - Even if s is 0.05, speedup cannot be more than 20
    - To get very large speedup, s should be tiny

Performance comparison

- Root of most debates and confusion
  - Consider three computers A, B, C and programs P1, P2
  - A executes P1 in 1 second and P2 in 1000 seconds
  - B executes P1 in 10 seconds and P2 in 100 seconds
  - C executes P1 in 20 seconds and P2 in 20 seconds
  - Which computer is better?
  - One possibility: calculate total time to execute P1 and P2, and compare them
  - Along the same line it is possible to report arithmetic mean of the total time e.g. in this case 0.5(t(P1) + t(P2))
  - On average A takes 500.5s, B takes 55s, C takes 20s to execute P1 and P2
  - Is it a fair comparison? What could be wrong?
    - Weighted AM is better?

Benchmarks

- Want to compare two processors by measuring their performance
  - Need some standardized set of programs
  - These are called benchmark programs
  - Different types of computers have different focus: needs different set of benchmarks
    - Performance metric for desktop/workstation is execution time (also known as response time)
    - Performance metric for servers is throughput (number of jobs done per unit time with a limit on response time per job)
    - Performance metric for embedded processors is also execution time with an emphasis on deadline and power consumption

Desktop benchmarks

- SPEC CPU is the standard
  - Provided by Standard Performance Evaluation Corporation (SPEC) *http://www.spec.org/*
  - Currently in the 6th generation: SPEC89, SPEC92, SPEC95, SPEC2000, SPEC2006, SPEC2017
  - Has two types of programs: integer and floating-point to stress different CPU units (C, C++, Fortran)
  - If you are introducing a new processor for desktop or uniprocessor workstation/server, you are supposed to report performance of all SPEC CPU 2017 programs
  - For graphics performance two available benchmarks: SPECviewperf and SPECapc
  - For Java run-time environment's performance: SPECjvm

SPEC CPU 2017 integer programs

- 10 applications covering a wide range
  - perl (popular scripting language)
  - gcc (part of GNU C compiler)
  - mcf (public transit scheduling)
  - omnetpp (discrete event simulation of ethernet network)
  - xalancbmk (XML processing)
  - x264 (video compression)
  - deepsjeng (AI alpha-beta tree search in chess)
  - leela (AI monte carlo tree search in go)
  - exchange2 (AI recursive solution generator in sudoku)
  - xz (general data compression)

SPEC CPU performance

- For each application, a computer's performance is measured by a number known as the SPEC ratio
  - SPEC provides the execution time of the application on a reference computer
  - SPEC ratio is the reference execution time provided by SPEC divided by the execution time of the application on the target computer
  - Further, the SPEC ratios of all the applications can be summarized by using a statistical sketch
    - One popular way is to report the geometric mean (GM) of the SPEC ratios
    - The computer having the largest geometric mean is the best for this set of applications
    - Why GM? Comparison is independent of the reference time (think of SPECratio(C1)/SPECratio(C2) where C1 and C2 are two computers that you want to compare)

Measuring computer power

- Logic gates are built using transistors
- When the output of a logic gate switches from 0 to 1 or 1 to 0, it generates current and charges or discharges the equivalent output capacitance of the gate
  - Expends $0.5*C*V^2$ energy for each transition where C is the equivalent output capacitance (depends on the fan-out of the gate) and V is the supply voltage
  - If number of such switching in the entire computer is N in a clock cycle, total energy expense per cycle is proportional to NCV^2

Measuring computer power

- Power consumption is energy per unit time which is proportional to NCV^2/cycle time or equivalently NCV^2f where f is the clock frequency
- Total energy expense of a program is NCV^2t where t is the total number of cycles needed to execute the program
- The energy expended in capacitive switching is known as the dynamic energy
- Over generations of computers, transistor size has dropped leading to a drop in V
- Frequency of designs has also increased

Measuring computer power

- Increase in f and drop in V may increase or decrease overall dynamic energy depending on the exact factors
  - Even if it increases, it increases at a smaller rate compared to rate of increase in f
- Drop in V has created a new problem called leakage current (also called static current)
  - Caused by transistors that do not switch off completely even when doing nothing
  - Creates a resistive path from supply to ground expending energy during inactive periods
  - Expends leakage or static energy and increases in magnitude with decreasing transistor size

Measuring computer power

- Dynamic and leakage (or static) energy are big problems
- Limits increase in clock frequency, limits complexity of logic, limits decrease in supply voltage with shrinking in transistor size
- Leakage is a bigger problem because it grows exponentially with lowering of supply voltage
- Many solutions today in computers
  - Dynamically varying voltage and frequency depending on need
  - Specialized circuits to arrest leakage current

Moore's law

- Number of transistors in a given area doubles every 18 to 24 months
  - Made possible by improved technology enabling shrinking of transistors
  - More transistors means more smartness in the chip and hence, more performance
    - Could design more sophisticate ALUs, for example
  - Moore's law has played a vital role in performance improvement of computers
  - Negative side: switching more transistors every clock cycle increases power consumption leading to more heating (needs sophisticated cooling solutions)
    - Moore's law is said to have "slowed down" these days because of this reason