Intro

Computers

Personal Computers:

* General purpose, variety of software
* Subject to cost/performance tradeoff

Server computers:

* Network based
* High capacity, performanc4e, reliability
* Small to building sized

Super computers

* Used for scientific and engineering calculations
* Highest capability but represent small fraction of the overall computer market

Embedded computers:

* Hidden as components of systems
* Stringent power/performance/cost constraints

Personal mobile device (PMD)

* Battery operated, connects to internet
* Smart phones, tablets, smart glasses

Below your program

Application software

* Written in high level language

System software

* Compiler
  + Translates HLL code to machine code
* Operation system: service code
  + Handles input/output
  + Manages memory and storage
  + Schedules tasks and resources

Hardware

* Processor, memory, I/O controllers

Determining performance

Algorithm

* Number of operations executed

Programming language, compiler, architecture

* Number of instructions executed per operation

Processor and memory system

* How fast instructions executed

I/O System (Includes OS)

* How fast I/O operations executed

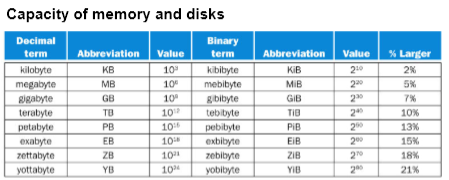
Technology, Performance, Power

Trends

Increased capacity and performance for reduced cost

Moore’s Law

* Every 1.5 years -> 2x transistors / chip

Terminology

Speed

* KHz, MHz, GHz
* KIPS, MIPS, GIPS

Inside Processor

Datapath :Performs operations on data

Control: Sequences datapath and memory

Cache memory: Small fast SRAM memory for immediate access to data

Storage

Volatile main memory

* Loses instructions and data when power off

Secondary memory

* Magnetic disk (HDD), flash memory, optical disk

Response Time and Throughput

Response time:

* How long it takes to do a task

Throughput

* Total work done per unit time
* Eg. tasks / hour

Relative performance

Define performance = 1 / time

“x is n time faster than y”

* PerformanceX / PerformanceY = TimeX / TimeY = n

Measuring time

Elapsed time

* Total time including all aspects
* Determines system performance

CPU Time

* Time spent processing
* Discounts I/O time, other jobs shares
* Comprises user CPU time and system CPU time

Different programs affected differently by CPU and system performance

CPU Clocking

Operation of digital hardware governed by constant-rate clock

Clock period / ycle time

* Duration of clock cycle
* Eg. 250ps = .25ns = 250 x 10-12 s

Clock frequency (CPU Speed)

* Cycles per second
* 4.0 GHz = 4000MHz = 4.0 x 109 Hz

Program CPU Time

Clock cycles = Instruction count \* cycles per instruction

CPU Time = CPU Clock Cycles \* Clock Cycle Time

= CPU Clock Cycles / Clock Rate

CPU Time = Instruction count \* CPI \* (1 / clock rate) = seconds

Performance improved by:

* Reducing program clock cycles
* Increasing clock rate

Instruction count

* Determined by program, ISA, compiler

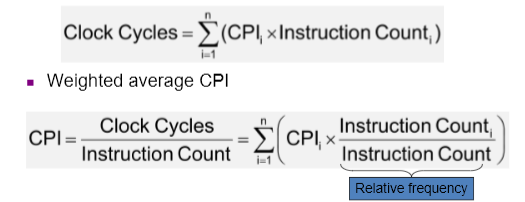
Average cycles per instruction

* Determined by hardware

Clocks per instruction (CPI)

If different instruction classes take different numbers of cycles…

* CPI of each class is specific and constant, determined by hardware
* Instruction count of each class of instruction in program can be obtained with certain tools



Principles of Arch Design

Make common case fast (90 / 10 rule)

* Most effective approach for performance improvement

Amdahl’s Law

* Law of diminishing returns

Speedup

* Achieved performance improvement over original

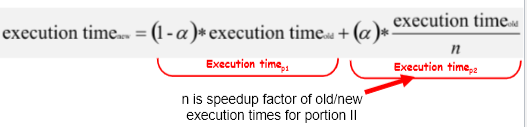
Principles of Arch Design

Execution time of any code has two portions:

* Portion 1: Not affected by enhancement
* Portion 2: Affected by enhancement
* α = % of code belonging to portion 2

Execution timeold = execution timep1 + execution timep2

As n→inf., execution timenew → (1-α) \* timeold



Pitfalls in performance evalulation

Using subset of performance equation as performance metric

* Clock rate, CPI, or IC only (or just two)

Using million instructions per second as metric

* Can’t be used to compare different ISAs
* Varies with programs on same computer
* Vary independent from performance

Benchmarking

* Test that measures performance of system or subsystem on well-defined task(s)
* Method of comparing performance of different computer architecture
* Method of comparing performance of different software
* Report performance data

Types

Real programs

* Input, output, options for running
* E.g. compilers, text processing software

Kernels

* Small, key pieces from real programs
* Eg. Livermore loops and linpack

Toy benchmarks

* 10-100 lines of code
* Produce result user already knows

Synthetic benchmarks

* Match an average exec. Profile

Pitfalls

Optimization on modern compilers affects results

Modification of sources produces different versions of benchmark

Many test only one aspect of system

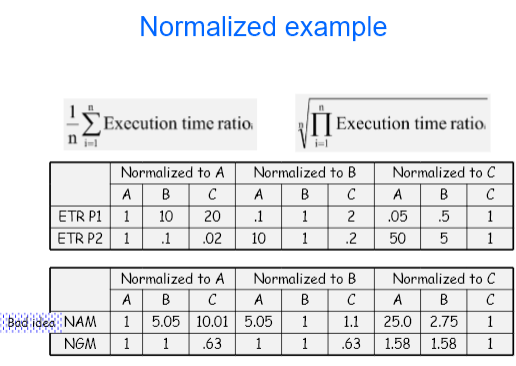
Compiler can recognize benchmark suite and loads hand-optimized algorithms for the test

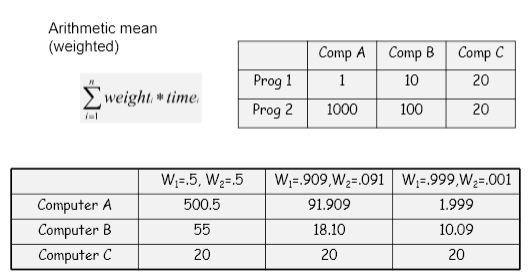
Perverse incentive: Vendors drove to improve benchmarks

Suites

Collection of benchmarks measuring performance of systems/subsystems with variety of application

* Weakness of one benchmark lessened by presence of others
* Some are kernels, others real programs

Mean



Arithmetic vs Geometric

Arithmetic

* Weighted average
* Pro: Propertional to overall execution time
* Cons:
  + Rigged easily
  + Cannot be normalized

Normalized Geometric mean:

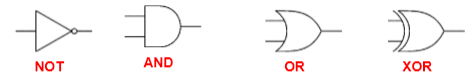
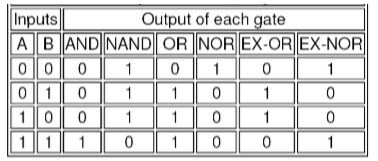
* Provides relative performance of machines to reference
* Pros:
  + Same results regardless of machine
* Cons:
  + Not proportional to overall execution time
  + Large % change in small overall time contributor can skew

Suggestion:

* Weight programs to frequency
* Use problem size to pre-normalize program execution time
* Combine approaches: Summary of simple means and relative performance to base machine

Combinational Logic

Logic Gates



Boolean Algebra:

* OR: + or OR
* AND: \* or AND
* NOT: ~
* XOR: 

Laws

* Commutative:
  + A+B=B+A
  + AB=BA
* AssosciatE:
  + (A + B) + C = A + (B + C)
  + (A B) C = A (B C)
* Distributive
  + A (B + C) = A B + A C
  + A + (B C) = (A + B) (A + C)
* Assorted:
  + 0 + A = A
  + 1A = A
  + 1 + A = 1
  + 0A = 0
  + ~A + A = 1
  + ~AA = 0
* Demorgan’s Theorem
  + ~(A + B) = ~A~B
  + ~(AB) = ~A + ~B

Literals, Minterms, SOP

A variable X has two literals: x and ~x

A logical product where each variable by at most one literal is:

* Product
* Product term or
* Term

Minterm is logical product of n literals where each variable occurs as exactly one literal

Canonical SOP is a logical sum of minterms where all minterms are different

Given A, B, C:

* 6 literals: A, B, C, ~A, ~B, ~C

Multiplexers

n-1 multiplexer: device that allows you to pick one of n inputs and direct it to an output

* Data input: n
* Control input: ceil\*(log2n)
* Output: 1
* E.g: 2-1 MUX: if c=0, z = x0, if c=1, z = x1

Sequential Logic

Sequential logic circuits

* One or more combinational logic blocks
* Have state in a feedback loop with logic
* Use clock
* Next state depends on inputs and present state
* Output depends on present state and perhaps input