Metrics and Evaluation

Introduction

1. In order to build better computers, we need to define "better"
    - Determining the right metrics for evaluating performance

Performance

1. Latency: Time between start and finish
2. Throughput: number/unit time
    - Throughput != 1 / Latency due to pipelining
    - Example: Cars on an assembly line
        - Entire process takes 4 hours with 20 steps
        - Latency = 4 hours
        - Throughput = 5 cars / hour

Latency and Throughput Quiz

1. Website with 2 servers
    - Order request assigned to a server
    - Server takes 1 millisecond to process an order
    - Server cannot do anything else while processing an order
    - Throughput of website = 2000 orders/second
    - Latency of website = 1 millisecond

Comparing Performance

1. "X is N times faster than Y"
    - Speedup = N -> Speed(X) / Speed(Y)
    - N = Throughput(X) / Throughput(Y)
        - Since higher latency is better
    - N = Latency(Y) / Latency(X)
        - Since lower latency is better

Performance Comparison Quiz 1

1. Old laptop can compress video in 4 hours
2. New laptop can compress video in 10 minutes
    - Speedup(new vs old) = 4 * 60 / 10 = 24
    - Speedup(old vs new) = 10 / 4 / 60 = 0.041666

Speedup

1. Speedup > 1 = Improved performance
    - Shorter execution time
    - Higher throughput
2. Speedup < 1 = Worse performance
    - Higher execution time
    - Lower throughput
3. Performance ~ Throughput
4. Performance ~ 1 / Latency

Measuring Performance

1. Performance = 1 / Execution time
    - On what workload? Potentially actual user workload, but...
        - Requires running many programs
        - Not representative of other users
        - How do we get workload data?
    - Instead, typically use a benchmark workload

Benchmarks

1. Programs and input data agreed upon by users for performance measurements
2. Benchmark suite
    - Multiple programs
    - Each representative of some type of application

Types of Benchmarks

1. Real applications
    - Most representative
    - Most difficult to set up (full OS, graphics card, hard drives)
    - Good for comparing actual machines
2. Kernels
    - Find most time-consuming part of an application
    - Isolate loop and use it for testing
    - Still too difficult in early stages of prototyping (no compiler)
    - Good for prototyping
3. Synthetic benchmarks
    - Behave similarly to kernels but simpler to compile
    - Good for design studies
4. Peak performance
    - Maximum number of instructions/second
    - Good for marketing

Performance Reporting Quiz

1. Which of these use code from real applications, but not all of the code?
    - Application kernel

Benchmark Standards

1. Benchmarking organization
    - Takes input from user groups, experts in academia, manufacturers
    - Creates a standard benchmark suite
2. Industry standard benchmarks
    - TPC: Databases, web servers, data mining, transaction processing
    - EEMBC: Embedded processing (cars, phones, etc)
    - SPEC: Engineering workstations and raw processors (not very I/O intensive)
        - GCC
        - Bwaves, LBM
        - PERL
        - Cactus ADM
        - Xalanc BMK
        - Calculix, Deall
        - Bzip

  - Go, Sjeng

Summarizing Performance

1. Average execution time
   - Can't simply average speedup for individual applications; not equivalent to average speedup overall
     - Instead, use geometric mean
     - Geometric mean = nth root(product(vals))
   - Simple arithmetic mean should not be used on ratios of times

Application	Computer X	Computer Y	Speedup
A	9 secs	18 secs	2
B	10 secs	7 secs	0.7
C	5 secs	11 secs	2.2
Average	8 secs	12 secs	1.5
Geometric	7.66 secs	11.15 secs	1.456

Speedup Averaging Quiz

1. Old laptop to new laptop
2. Homework formatting application gives speedup of 2
3. Virus scan application gives speedup of 8
4. Overall speedup = sqrt(2 * 8) = 4

Iron Law of Performance

1. CPU time = # instructions in program * cycles per instruction * clock cycle time
   - CPU time = (insts / program) * (cycles / insts) * (seconds / cycle)
2. Instructions in program
   - Algorithm
   - Compiler
   - Instruction set
3. Cycles per instruction
   - Instruction set
   - Processor design
4. Clock cycle time
   - Processor design
   - Circuit design
   - Transistor physics
5. Computer architecture affects the instruction set and processor design
   - Complex instructions: Fewer instructions, more cycles per instruction
   - Simpler instructions: More instructions, fewer cycles per instruction
   - Short clock cycle: More cycles per instruction
   - Longer clock cycle: Fewer cycles per instruction
   - Goal is to balance

Iron Law Quiz 1

1. Program executes 3 billion instructions
2. Processor spends 2 cycles per instruction
3. Processor clock speed is 3 GHz
4. Execution time = 3e9 * 2 * (1/3e9) = 2 seconds

Iron Law of Unequal Instruction Times

1. Execution time of instructions can vary
2. CPU time = sum(ICi * CPIi) * (time / cycle)

Iron Law Quiz 2

1. Program executes 50 billion instructions
   - 10 billion are branches (CPI=4)
   - 15 billion are loads (CPI=2)
   - 5 billion are stores (CPI=3)
   - The rest are integer add (CPI=1)
2. Clock at 4 GHz
3. Execution time = (4 * 10e9 + 2 * 15e9 + 3 * 5e9 + 20 * 1e9) / 4e9 = 26.25

Amdahl's Law

1. Used to speed up part of the program (only some instructions)
2. What is overall speedup?
   - Speedup = 1 / ((1 - FRACenh) + (FRACenh / SPEEDUPenh))
   - FRACenh = percentage of original execution time affected by enhancement

Amdahl's Law Quiz 1

1. Program has 50 billion instructions
2. Clock frequency is 2 GHz
3. Improve branch instruction from 4 to 2 CPI
4. Can't just plug into Amdahl's law since these are percentages of instructions, not execution times
   - (0.4 * 1 + 0.2 * 4 + 0.3 * 2 + 0.1 * 3) * 1e9 / 2e9 = 1.05 (before enhancement)
   - (0.4 * 1 + 0.2 * 2 + 0.3 * 2 + 0.1 * 3) * 1e9 / 2e9 = 0.85 (after enhancement)
   - 1.05 / 0.85 = 1.24

Inst Type	% in program	CPI
Interrupt	40%	1
Branch	20%	4
Load	30%	2
Store	10%	3

Amdahl's Law Implications

1. Enhancement 1: Speedup of 20 on 10% of time
   - 1 / ((1 - 0.1) + (0.1 / 20)) = 1.105
   - Even if infinite speedup, overall speedup is only 1.111
2. Enhancement 2: Speedup of 1.6 on 80% of time
   - 1 / ((1 - 0.8) + (0.8 / 1.6)) = 1.43
3. Conclusion: Make common case fast
   - Better to affect lots of code a little than a small amount of code a lot

Amdahl's Law Quiz 2

1. Program has 50 billion instructions
2. Clock frequency is 2 GHz
3. Possible improvements
   - Branch CPI 4 -> 3

- Increase clock frequency 2 -> 2.3
- Store CPI 3 -> 2

4. Which is best?
   - Branch: 1 / ((1 - 0.2) + 0.2 / (4 / 3)) = 1.05
   - Clock: 1 / ((1 - 1) + 1 / (2.3 / 2)) = 1.15
   - Store: 1 / ((1 - 0.1) + 0.1 / (3 / 2)) = 1.034

Inst Type	% in program	CPI
Interrupt	40%	1
Branch	20%	4
Load	30%	2
Store	10%	3

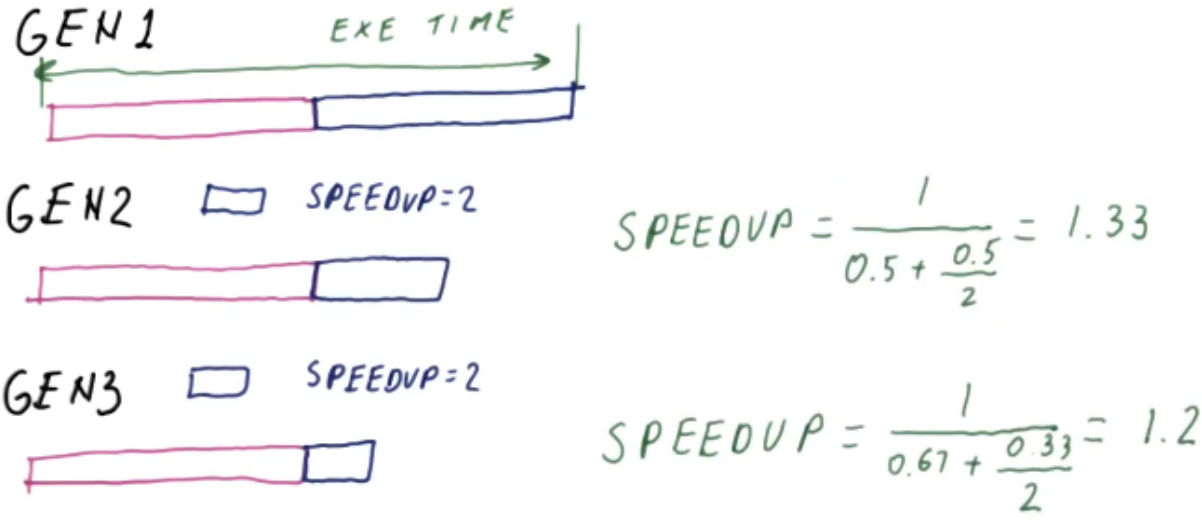
Lhadma's Law

1. Amdahl: Make common case fast
2. Lhadma: Do not mess up uncommon case too badly
3. Example:
   - Improvement of 2x on 90%
   - Slow down the rest by 10x
   - Speedup = 1 / (0.1/0.1 + 0.9/2) = 1 / 1.45 = 0.7 so overall speed is worse

Diminishing Returns

1. Don't go overboard improving the same thing
   - As percentage of execution time decreases, overall speedup also decreases
   - Additionally, once the easier optimizations are complete, more time is required for further improvements

---



Diminishing Returns

Conclusion

1. Overall performance improvement requires balancing improvements to an aspect without detriment to another aspect