ECE/CS 552: Course Overview & Context

Lecture notes based in part on slides created by Mark Hill, David Wood, Guri Sohi, John Shen and Jim Smith

552 In Context

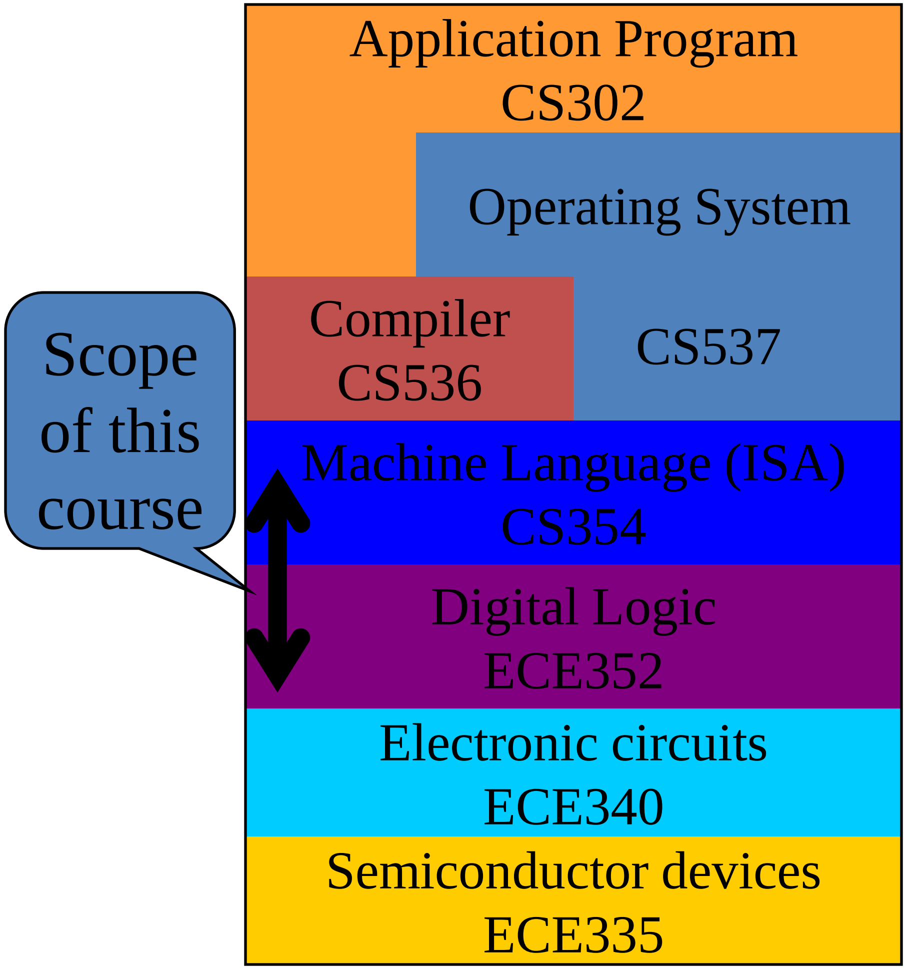
- Prerequisites
  - 252/352 – gates, logic, memory, organization
  - 252/354 – high-level language down to machine language interface or instruction set architecture (ISA)
- This course – 552 – puts it all together
  - Implement the logic that provides ISA interface
  - Must implement datapath and control
  - You will understand…no mystery
  - Manage tremendous complexity with abstraction

Why Take 552?

- To become a computer designer
  - Alumni of this class helped design your computer
- To learn what is *under the hood* of a computer
  - Innate curiosity
  - To write better code/applications
  - To write better system software (O/S, compiler, etc.)
- Because it is intellectually fascinating!
  - What is the most complex man-made device?

Abstraction and Complexity

- Abstraction helps us manage complexity
- Complex interfaces
  - Specify what to do
  - Hide details of how
- Goal: remove mystery



Computer Architecture

- Exercise in engineering tradeoff analysis
  - Find the fastest/cheapest/power-efficient/etc. solution
  - Optimization problem with 100s of variables
- All the variables are changing
  - At non-uniform rates
  - With inflection points
  - Only one guarantee: Today's right answer will be wrong tomorrow
- Two high-level effects:
  - Technology push
  - Application Pull

Technology Push

- What do these two intervals have in common?
  - 1947-1999 (53 years)
  - 2000-2001 (2 years)
- Answer: Equal progress in processor speed!
- The power of exponential growth!
- Driven by Moore's Law
  - Device per chips doubles every 18-24 months
- Computer architects work to turn the additional resources into speed/power savings/functionality!

Some History

Date	Event	Comments
1939	First digital computer	John Atanasoff (UW PhD ’30)
1947	1st transistor	Bell Labs
1958	1st IC	Jack Kilby (MSEE ’50) @TI Winner of 2000 Nobel prize
1971	1st microprocessor	Intel
1974	Intel 4004	2300 transistors
1978	Intel 8086	29K transistors
1989	Intel 80486	1.M transistors, pipelined
1995	Intel Pentium Pro	5.5M transistors
2005	Intel Montecito	1B transistors

Performance Growth

Unmatched by any other industry !

Doubling every 18 months (1982-1996): 800x
- Cars travel at 44,000 mph and get 16,000 mpg
- Air travel: LA to NY in 22 seconds (MACH 800)
- Wheat yield: 80,000 bushels per acre

- Doubling every 24 months (1971-1996): 9,000x
  - Cars travel at 600,000 mph, get 150,000 mpg
  - Air travel: LA to NY in 2 seconds (MACH 9,000)
  - Wheat yield: 900,000 bushels per acre

Technology Push

- Technology advances at varying rates
  - E.g. DRAM capacity increases at 60%/year
  - But DRAM speed only improves 10%/year
  - Creates gap with processor frequency!
- Inflection points
  - Crossover causes rapid change
  - E.g. enough devices for multicore processor (2001)
- Current issues causing an “inflection point”
  - Power consumption
  - Reliability
  - Variability

Application Pull

- Corollary to Moore's Law:
  Cost halves every two years

  In a decade you can buy a computer for less than its sales tax today. –Jim Gray

- Computers cost-effective for
  - National security – weapons desi
  - Enterprise computing –
  - Departmental … sign
  - Pe … web
  - P … drug labels

That was the old days….Now computers cost effective for even the most trivial of applications. What are the future applications? That is your job to figure out.

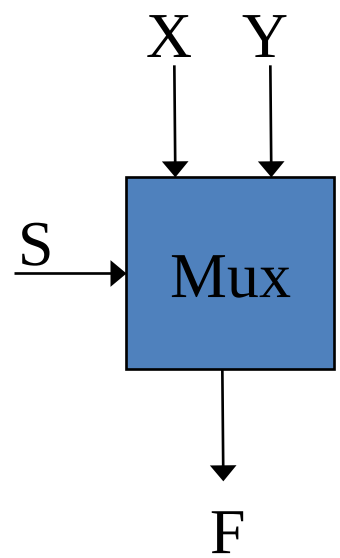
Abstraction

- Difference between interface and implementation
  - Interface: WHAT something does
  - Implementation: HOW it does so

- ***Career note****…Those who stay at the higher level with WHAT and don't get too distracted by HOW have more successful long term engineering careers.*

Abstraction, E.g.

- 2:1 Mux (352)
- Interface

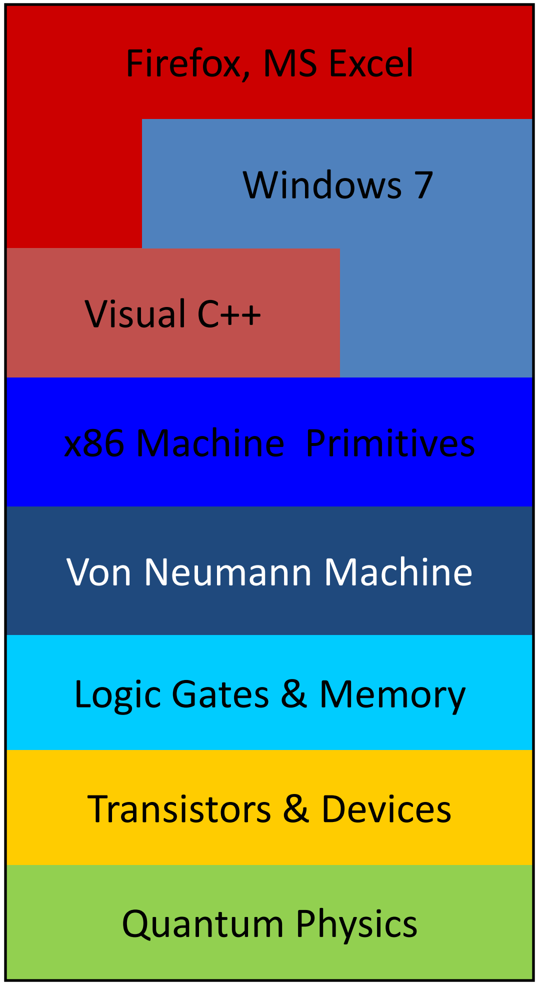


S	F
0	X
1	Y

- Implementations
  – Gates (fast or slow), pass transistors

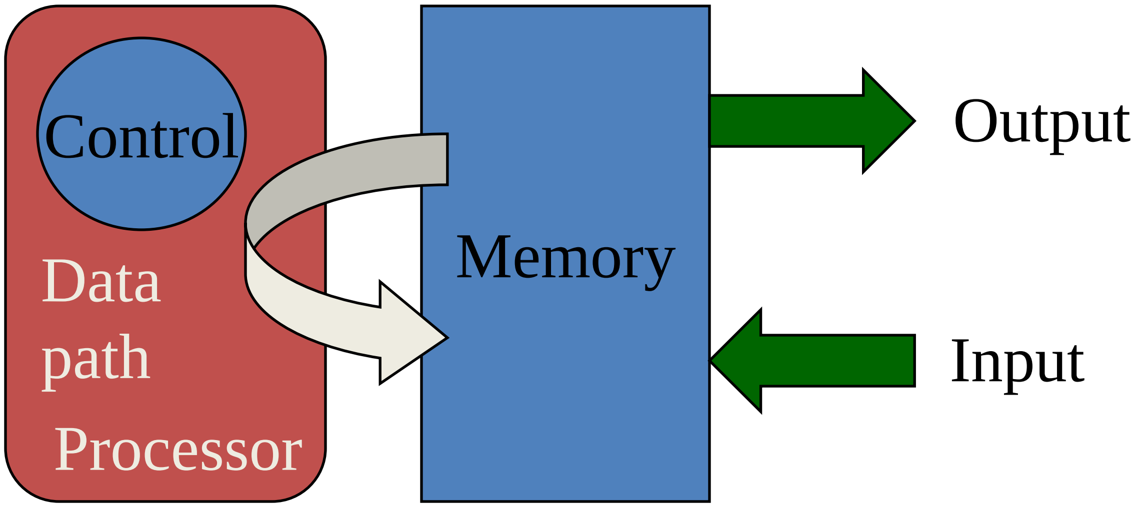
What's the Big Deal?

- Tower of abstraction
- Complex interfaces implemented by layers below
- Abstraction hides detail
- Hundreds of engineers build one product
- Complexity unmanageable otherwise



Basic Division of Hardware

- In space (vs. time)



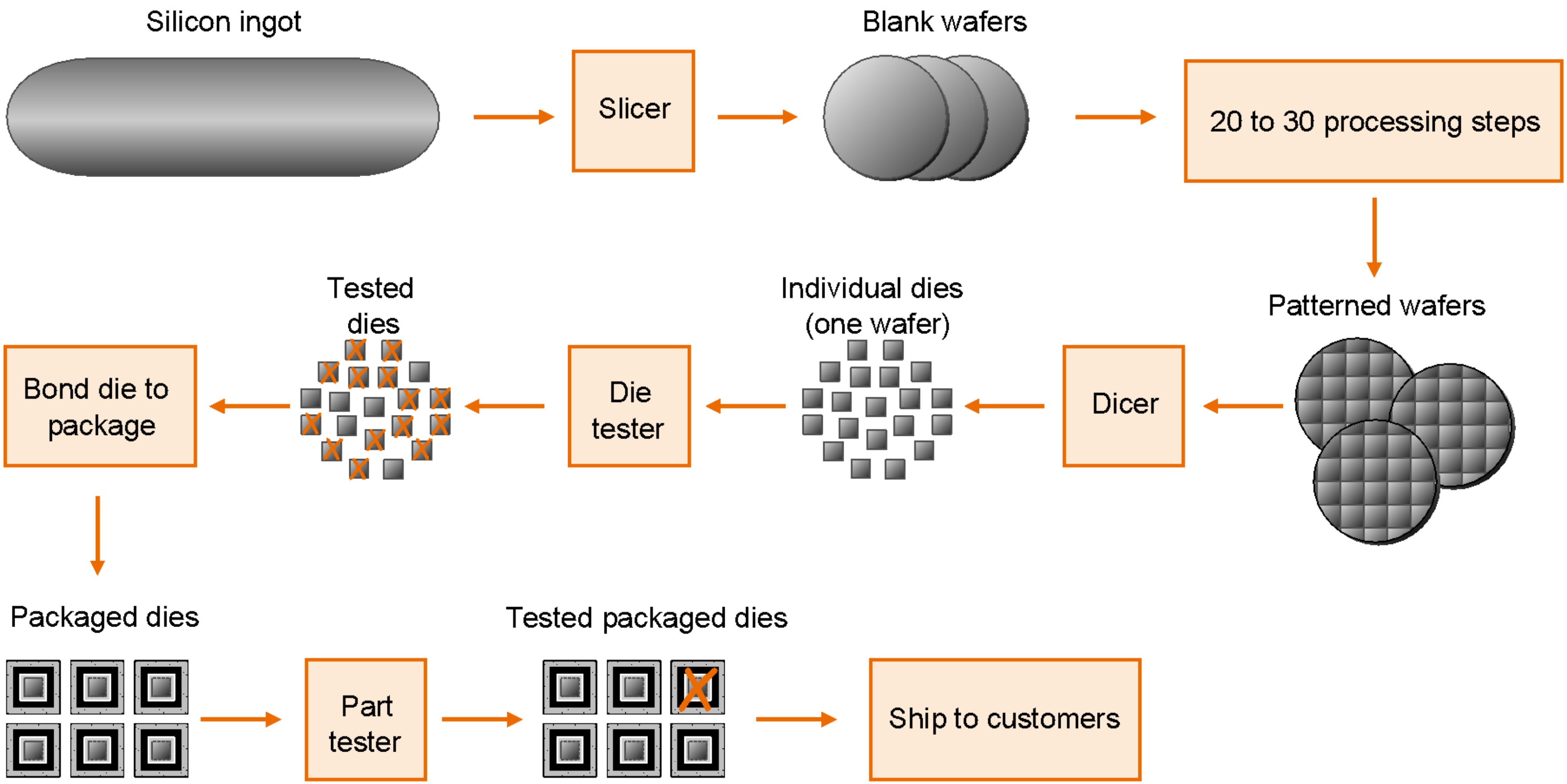
Basic Division of Hardware

- In time (vs. space)
  - Fetch instruction from memory **add r1, r2, r3**
  - Decode the instruction – what does this mean?
  - Read input operands read r2, r3
  - Perform operation add
  - Write results write to r1
  - Determine the next instruction pc := pc + 4

Building Computer Chips

- Complex multi-step process
  - Slice silicon ingots into wafers
  - Process wafers into patterned wafers
  - Dice patterned wafers into dies
  - Test dies, select good dies
  - Bond to package
  - Test parts
  - Ship to customers and make money

Building Computer Chips



Performance vs. Design Time

- Time to market is critically important
- E.g., a new design may take 3 years
  - It will be 3 times faster
  - But if technology improves 50%/year
  - In 3 years $1.5^3 = 3.38$
  - So the new design is worse!

    (unless it also employs new technology)

Bottom Line

- Designers must know BOTH software and hardware
- Both contribute to layers of abstraction
- IC costs and performance
- Compilers and Operating Systems

ECE/CS 552: Performance and Cost

Lecture notes based in part on slides created by Mark Hill, David Wood, Guri Sohi, John Shen and Jim Smith

Forecast

- Time and performance
- Iron Law
- MIPS and MFLOPS
- Which programs and how to average
- Amdahl’s law

Performance and Cost

- Which of the following airplanes has the best performance?

Airplane	Passengers	Range (mi)	Speed (mph)
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

- How much faster is the Concorde vs. the 747
- How much bigger is the 747 vs. DC-8?

Performance and Cost

- Which computer is fastest?
- Not so simple
  - Scientific simulation – FP performance
  - Program development – Integer performance
  - Database workload – Memory, I/O

Performance of Computers

- Want to buy the fastest computer for what you want to do?
  - Workload is all-important
  - Correct measurement and analysis
- Want to design the fastest computer for what the customer wants to pay?
  - Cost is an important criterion

Defining Performance

- What is important to whom?
- Computer system user
  - Minimize elapsed time for program:

    $t_{resp} = t_{end} - t_{start}$
  - Called response time
- Computer center manager
  - Maximize completion rate = #jobs/second
  - Called throughput

Response Time vs. Throughput

- Is throughput = 1/avg. response time?
  - Only if NO overlap
  - Otherwise, throughput > 1/avg. response time
  - E.g. a lunch buffet – assume 5 entrees
  - Each person takes 2 minutes/entrée
  - Throughput is 1 person every 2 minutes
  - BUT time to fill up tray is 10 minutes
  - Why and what would the throughput be otherwise?
    - 5 people simultaneously filling tray (overlap)
    - Without overlap, throughput = 1/10

What is Performance for us?

- For computer architects
  - CPU time = time spent running a program
- Intuitively, bigger should be faster, so:
  - Performance = 1/X time, where X is response, CPU execution, etc.
- Elapsed time = CPU time + I/O wait
- We will concentrate on CPU time

Improve Performance

- Improve (a) response time or (b) throughput?
  - Faster CPU
    - Helps both (a) and (b)
  - Add more CPUs
    - Helps (b) and perhaps (a) due to less queueing

Performance Comparison

- Machine A is n times faster than machine B iff

  perf(A)/perf(B) = time(B)/time(A) = n

- Machine A is x% faster than machine B iff

  perf(A)/perf(B) = time(B)/time(A) = 1 + x/100

- E.g. time(A) = 10s, time(B) = 15s
  - 15/10 = 1.5 => A is 1.5 times faster than B
  - 15/10 = 1.5 => A is 50% faster than B

Breaking Down Performance

- A program is broken into instructions
  - H/W is aware of instructions, not programs
- At lower level, H/W breaks instructions into cycles
  - Lower level state machines change state every cycle
- For example:
  - 1GHz Snapdragon runs 1000M cycles/sec, 1 cycle = 1ns
  - 2.5GHz Core i7 runs 2.5G cycles/sec, 1 cycle = 0.25ns

Iron Law

$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) **(CPI)** **(cycle time)**

Architecture --> Implementation --> Realization

Compiler Designer **Processor Designer** **Chip Designer**

Iron Law

- Instructions/Program
  - Instructions executed, not static code size
  - Determined by algorithm, compiler, ISA
- Cycles/Instruction
  - Determined by ISA and CPU organization
  - Overlap among instructions reduces this term
- Time/cycle
  - Determined by technology, organization, clever circuit design

Our Goal

- Minimize time which is the product, NOT isolated terms
- Common error to miss terms while devising optimizations
  - E.g. ISA change to decrease instruction count
  - BUT leads to CPU organization which makes clock slower
- Bottom line: terms are inter-related

Other Metrics

- MIPS and MFLOPS
- MIPS = instruction count/(execution time x 10^6)

  = clock rate/(CPI x 10^6)
- But MIPS has serious shortcomings

Problems with MIPS

- E.g. without FP hardware, an FP op may take 50 single-cycle instructions
- With FP hardware, only one 2-cycle instruction

- Thus, adding FP hardware:
  - CPI increases (why?) 50/50 => 2/1
  - Instructions/program decreases (why?) 50 => 1
  - Total execution time decreases 50 => 2
- BUT, MIPS gets worse! 50 MIPS => 2 MIPS

Problems with MIPS

- Ignores program
- Usually used to quote peak performance
  - Ideal conditions => guaranteed not to exceed!
- When is MIPS ok?
  - Same compiler, same ISA
  - E.g. same binary running on AMD Jaguar, Intel Core i7
  - Why? Instr/program is constant and can be factored out

Other Metrics

- MFLOPS = FP ops in program/(execution time x 10^6)
- Assuming FP ops independent of compiler and ISA
  - Often safe for numeric codes: matrix size determines # of FP ops/program
  - However, not always safe:
    - Missing instructions (e.g. FP divide)
    - Optimizing compilers
- Relative MIPS and normalized MFLOPS
  - Adds to confusion

Rules

- Use ONLY Time
- Beware when reading, especially if details are omitted
- Beware of Peak
  - “Guaranteed not to exceed”

Iron Law Example

- Machine A: clock 1ns, CPI 2.0, for program x
- Machine B: clock 2ns, CPI 1.2, for program x
- Which is faster and how much?

  Time/Program = instr/program x cycles/instr x sec/cycle

  Time(A) = N x 2.0 x 1 = 2N

  Time(B) = N x 1.2 x 2 = 2.4N

  Compare: Time(B)/Time(A) = 2.4N/2N = 1.2
- So, Machine A is 20% faster than Machine B for this program

Iron Law Example

Keep clock(A) @ 1ns and clock(B) @2ns

For equal performance, if CPI(B)=1.2, what is CPI(A)?

$\text{Time(B)/Time(A)} = 1 = (\cancel{N}x2x1.2)/(\cancel{N}x1xCPI(A))$

CPI(A) = 2.4

Iron Law Example

- Keep CPI(A)=2.0 and CPI(B)=1.2
- For equal performance, if clock(B)=2ns, what is clock(A)?

Time(B)/Time(A) = 1 = ($\cancel{N}$ x 2.0 x clock(A))/($\cancel{N}$ x 1.2 x 2)

clock(A) = 1.2ns

Summary

- Time and performance: Machine A n times faster than Machine B
  - Iff Time(B)/Time(A) = n
- Iron Law: Performance = Time/program =

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) **(CPI)** **(cycle time)**

- Other Metrics: MIPS and MFLOPS
  - Beware of peak and omitted details

ECE/CS 552: Benchmarks, Means and Amdahl's Law

© Prof. Mikko Lipasti

Lecture notes based in part on slides created by Mark Hill, David Wood, Guri Sohi, John Shen and Jim Smith

Which Programs

- Execution time of what program?
- Best case – you always run the same set of programs
  – Port them and time the whole workload
- In reality, use benchmarks
  – Programs chosen to measure performance
  – Predict performance of actual workload
  – Saves effort and money

Representative? Honest? Benchmarketing…

How to Average

	Machine A	Machine B
Program 1	1	10
Program 2	1000	100
Total	1001	110

- One answer: for total execution time, how much faster is B?

  1001 / 110 = 9.1x

How to Average

- Another: arithmetic mean (same result)
- Arithmetic mean of times:

$$\left\{\sum_{i=1}^{n} time(i)\right\} \times \frac{1}{n}$$

- AM(A) = 1001/2 = 500.5
- AM(B) = 110/2 = 55
- Speedup: 500.5/55 = 9.1x
- Valid only if programs run equally often, so use weighted arithmetic mean:

$$\left\{\sum_{i=1}^{n} \left(weight(i) \times time(i)\right)\right\} \times \frac{1}{n}$$

Other Averages

- E.g., 30 mph for first 10 miles, then 90 mph for next 10 miles, what is average speed?
- Average speed = (30+90)/2 **WRONG**
- Average speed = total distance / total time

= (20 / (10/30 + 10/90))

= 45 mph

Harmonic Mean

- Harmonic mean of rates = $$\frac{n}{\left\{\sum_{i=1}^{n} \frac{1}{rate(n)}\right\}}$$
- Use HM if forced to start and end with rates (e.g. reporting MIPS or MFLOPS)
- Why?
  - Rate has time in denominator
  - Mean should be proportional to inverse of sums of time (not sum of inverses)
  - See: J.E. Smith, “Characterizing computer performance with a single number,” CACM Volume 31 , Issue 10 (October 1988), pp. 1202-1206.

Dealing with Ratios

	Machine A	Machine B
Program 1	1	10
Program 2	1000	100
Total	1001	110

- If we take ratios with respect to machine A

	Machine A	Machine B
Program 1	1	10
Program 2	1	0.1
Average	1	5.05

Dealing with Ratios

- Avg. wrt. machine A: A is 1, 5.05
- If we take ratios with respect to machine B

	Machine A	Machine B
Program 1	0.1	1
Program 2	10	1
Average	5.05	1

- Can't both be true!!!
- Don't use arithmetic mean on ratios!

Geometric Mean

- Use geometric mean for ratios
- Geometric mean of ratios = $\sqrt[n]{\prod_{i=1}^{n} ratio(i)}$
- Independent of reference machine
- In the example, GM for machine a is 1, for machine B is also 1
  - Normalized with respect to either machine

But…

- GM of ratios is not proportional to total time
- AM in example says machine B is 9.1 times faster
- GM says they are equal
- If we took total execution time, A and B are equal only if
  - Program 1 is run 100 times more often than program 2
- Generally, GM will mispredict for three or more machines

Summary

- Use AM for times
- Use HM if forced to use rates
- Use GM if forced to use ratios

- Best of all, use unnormalized numbers to compute time

Benchmarks: SPEC2000

- System Performance Evaluation Cooperative
  - Formed in 80s to combat benchmarketing
  - SPEC89, SPEC92, SPEC95, SPEC2000, SPEC2006
- 12 integer and 14 floating-point programs
  - Sun Ultra-5 300MHz reference machine has score of 100
  - Report GM of ratios to reference machine

Benchmarks: SPEC CINT2000

Benchmark	Description
164.gzip	Compression
175.vpr	FPGA place and route
176.gcc	C compiler
181.mcf	Combinatorial optimization
186.crafty	Chess
197.parser	Word processing, grammatical analysis
252.eon	Visualization (ray tracing)
253.perlbmk	PERL script execution
254.gap	Group theory interpreter
255.vortex	Object-oriented database
256.bzip2	Compression
300.twolf	Place and route simulator

Benchmarks: SPEC CFP2000

Benchmark	Description
168.wupwise	Physics/Quantum Chromodynamics
171.swim	Shallow water modeling
172.mgrid	Multi-grid solver: 3D potential field
173.applu	Parabolic/elliptic PDE
177.mesa	3-D graphics library
178.galgel	Computational Fluid Dynamics
179.art	Image Recognition/Neural Networks
183.equake	Seismic Wave Propagation Simulation
187.facerec	Image processing: face recognition
188.ammp	Computational chemistry
189.lucas	Number theory/primality testing
191.fma3d	Finite-element Crash Simulation
200.sixtrack	High energy nuclear physics accelerator design
301.apsi	Meteorology: Pollutant distribution

Benchmark Pitfalls

- Benchmark not representative
  - Your workload is I/O bound, SPEC is useless
- Benchmark is too old
  - Benchmarks age poorly; benchmarketing pressure causes vendors to optimize compiler, hardware, software to match benchmarks
  - Need to be periodically refreshed

Amdahl’s Law

- Motivation for optimizing common case
- Speedup = old time / new time = new rate / old rate
- Let an optimization speed fraction *f* of time by a factor

Math: If f is small, s will have limited impact.

New_t ime + (f/s) x old_time

Speed ew_time

Speed 1-f) x old_time + (f/s) x old_time)

$$Speedup = \frac{[(1-\cancel{f})+\cancel{f}]\times \cancel{oldtime}}{[(1-f)\times \cancel{oldtime}]+\frac{f}{s}\times \cancel{oldtime}}$$

$$= \frac{1}{1-f+\frac{f}{s}}$$

Amdahl's Law Example

- Your boss asks you to improve performance by:
  - Improve the ALU used 95% of time by 10%
  - Improve memory pipeline used 5% of time by 10x

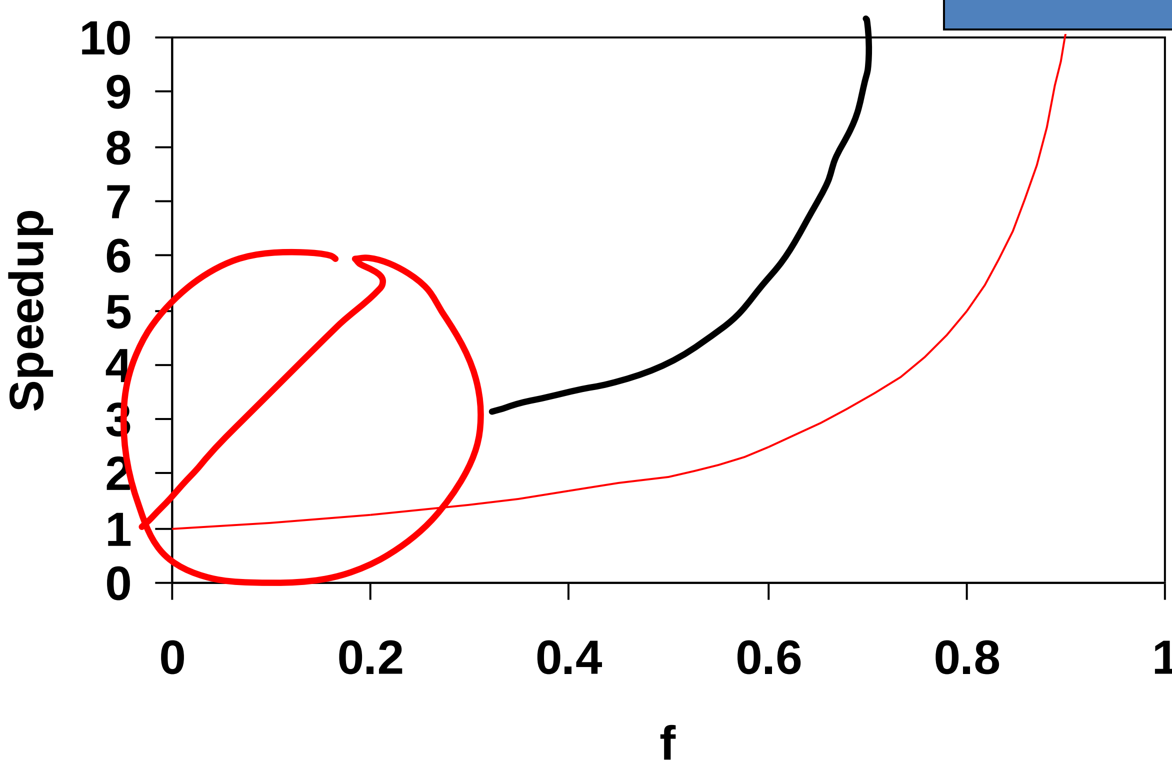
f	s	Speedup
95%	1.10	1.094
5%	10	1.047
5%	∞	1.052

$$Speedup = \frac{1}{1 - f + \frac{f}{s}}$$

Amdahl's Law: Limit

- Make common case fast:

$$\lim_{s \to \infty} \frac{1}{1 - f + \frac{f}{s}} = \frac{1}{1 - f}$$



Amdahl's Law: Limit

- Consider uncommon case!
- If (1-f) is nontrivial
  - Speedup is limited!

$$\lim_{s \to \infty} \frac{1}{1 - f + \frac{f}{s}} = \frac{1}{1-f}$$

- Particularly true for exploiting parallelism in the large, where large s is not cheap
  - GPU with e.g. 1024 processors (shader cores)
  - Parallel portion speeds up by s (1024x)
  - Serial portion of code (1-f) limits speedup

E.g. 10% serial portion: 1/0.1 = 10x speedup with 1000 cores

Summary

- Benchmarks: SPEC2000
- Summarize performance:
  - AM for time
  - HM for rate
  - GM for ratio
- Amdahl's Law:

$$Speedup = \frac{1}{1 - f + \frac{f}{s}}$$