CSE 548: Computer Systems Architecture

Metrics: Performance and Power

Spring 2017

Luis Ceze (Instructor)

Thierry Moreau (TA)

How do we measure computer *performance*?

Performance

- Two common measures
  - Latency (how long to do something)
    - Also called response time and execution time
  - Throughput (how *often* can it do something)
- Example of car assembly line
  - Takes 6 hours to make a car: *latency is 6 hours*
  - A car leaves every 5 minutes: *throughput is 12 cars per hour*
  - How can Throughput > 1/Latency

Comparing Performance

- Latency: "X is n times faster than Y"

$$\frac{\text{Execution time}_{Y}}{\text{Execution time}_{X}} = n$$

- Throughput: "Throughput of X is n times that of Y"

$$\frac{\text{Tasks per unit time}_{X}}{\text{Tasks per unit time}_{Y}} = n$$

What is the best way to choose system X versus Y?

If Only it Were That Simple

- "X is n times faster than Y *on A*"

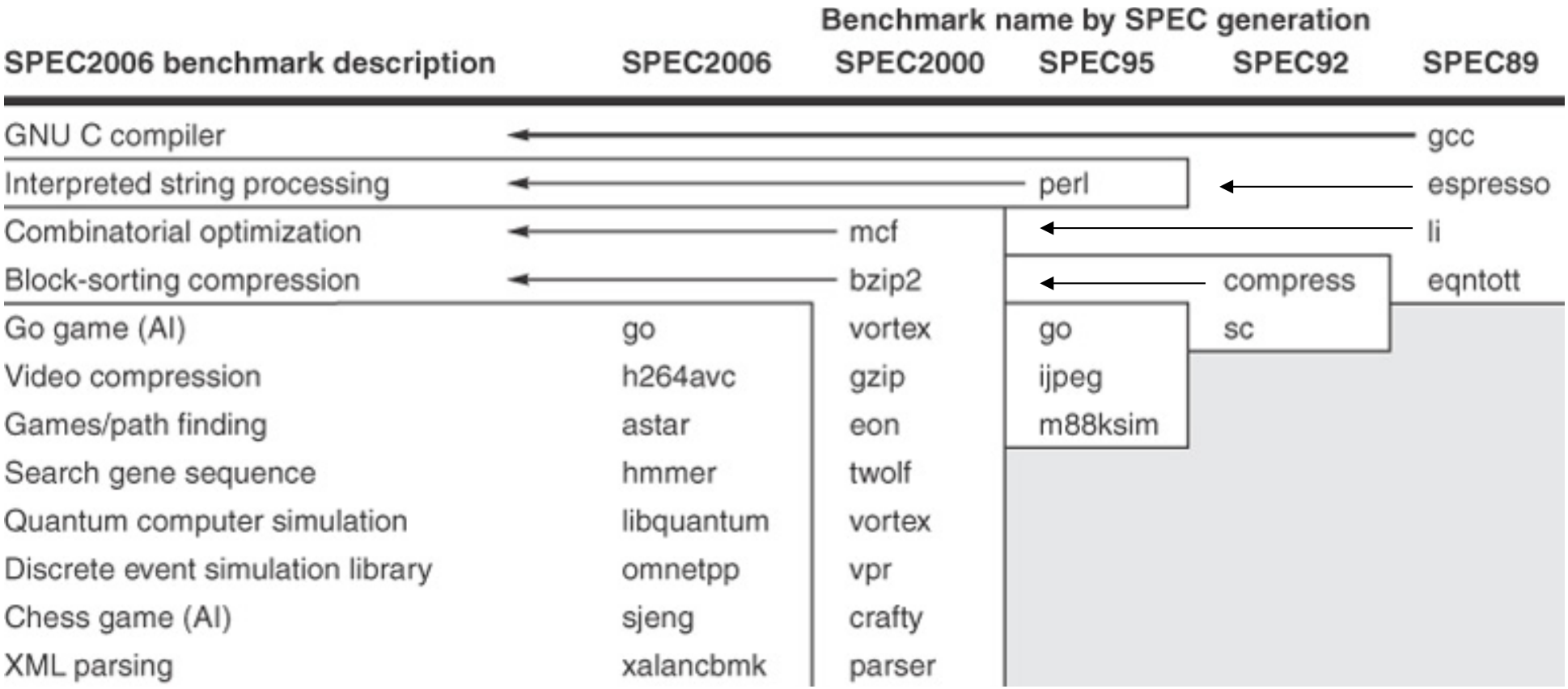
$$\frac{\text{Execution time of app A on machine Y}}{\text{Execution time of app A on machine X}} = \text{n}$$

- *But what about different applications*
  (or even parts of the same application)
  - X is 10 times faster than Y on A, and 1.5 times on B, but Y is 2 times faster than X on C, and 3 times on D, and…

Benchmarks

- Real applications and application suites
  - E.g., SPEC CPU2000, SPEC2006, TPC-C, TPC-H, Rodinia (GPUs), MiBench (embedded/mobile), GraphBench, …
- Kernels
  - "Representative" parts of real applications
  - Easier and quicker to set up and run
  - Often not really representative of the entire app
- Toy programs, synthetic benchmarks, etc.
  - In general, not very useful for reporting
  - Sometimes used to test/stress specific functions/features
    - E.g., memory bandwidth.

SPEC CPU (integer)



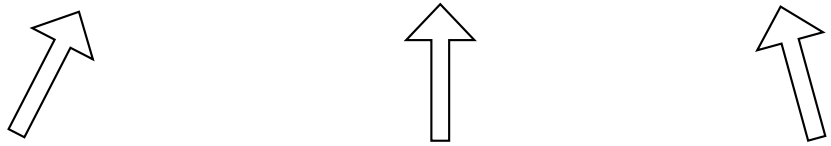
"Representative" applications keeps growing with time!

CPU Performance Equation (1)

$$\text{CPU time} = \text{CPU Clock Cycles} \times \text{Clock cycle time}$$

$$\text{CPU time} = \text{Instruction Count} \times \text{Cycles Per Instruction} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$



What affects each of these?

CPU Performance Equation (1)

$$\text{CPU time} = \text{CPU Clock Cycles} \times \text{Clock cycle time}$$

$$\text{CPU time} = \text{Instruction Count} \times \text{Cycles Per Instruction} \times \text{Clock cycle time}$$

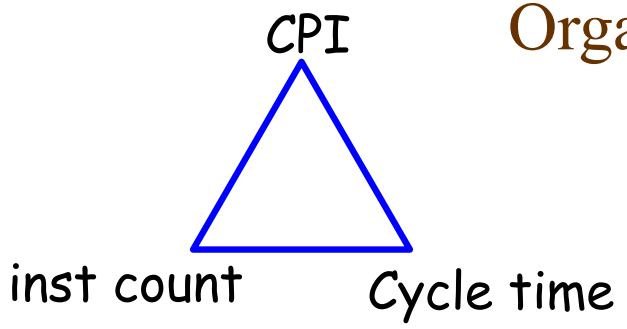
$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

ISA, Compiler Technology

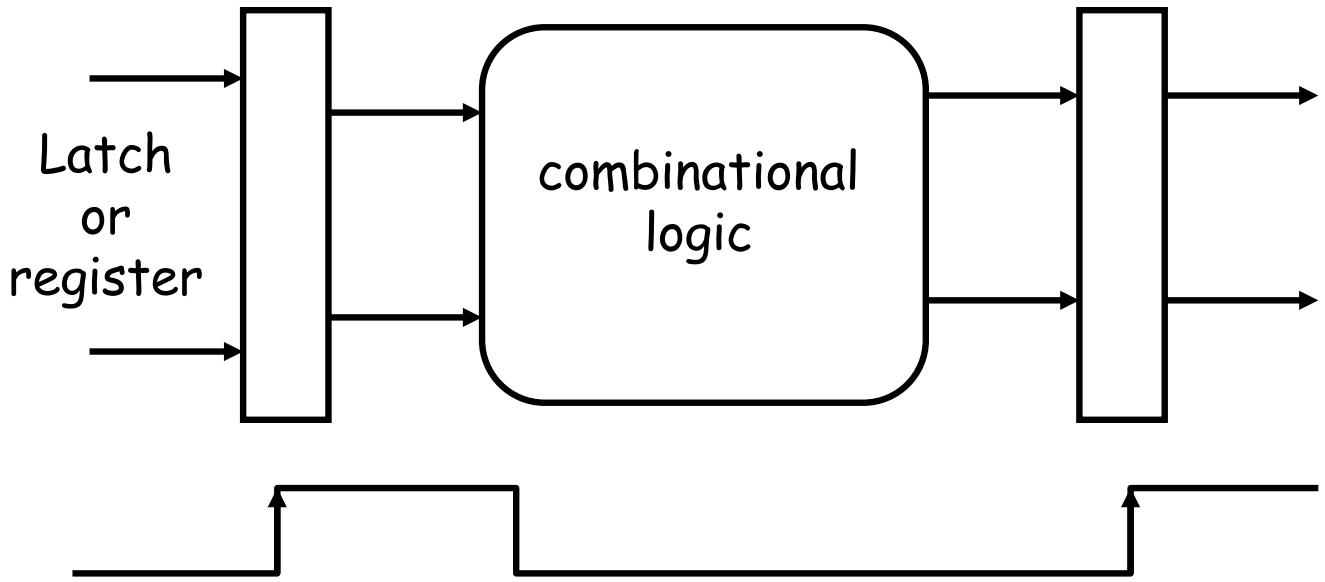
Organization, ISA

Hardware Technology, Organization

"Iron Law of Performance"



What's a Clock Cycle?



- Old days: 10+ levels of gates
- Today: determined by numerous time-of-flight issues + gate delays
  - clock propagation, wire lengths, repeaters
  - Can get *super complicated* -- design tools!!
- ~ 8-12 classic gate delays common

Example

- Program takes 33 billion instructions to run
- CPU processes insts at 2 cycles per inst
- Clock speed of 3GHz

Example

- Program takes 33 billion instructions to run
- CPU processes insts at 2 cycles per inst
- Clock speed of 3GHz

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

Sometimes clock cycle time given instead (ex. cycle = 333 ps)

IPC sometimes used instead of CPI

= 22 seconds

CPU Performance Equation (2)

$$\text{CPU time} = \text{CPU Clock Cycles} \times \text{Clock cycle time}$$

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

For each kind of instruction

How many instructions of this kind are there in the program

How many cycles it takes to execute an instruction of this kind

Calculating CPI

- Computed from instruction mix and CPI of each instruction type
- Very important quantitative metric

$$CPI = \sum_{i=1}^{N} \frac{IC_i}{IC} CPI_i$$

ALU	50%	1
Branches	15%	2
Loads	20%	2
Stores	15%	1

$$CPI = .5 \times 1 + .15 \times 2 + .2 \times 2 + .15 \times 1 = 1.35$$

Which processor would you buy?

- Processor A: CPI = 2, clock = 5 GHz
- Processor B: CPI = 1, clock = 3 GHz

Which processor would you buy?

- Processor A: CPI = 2, clock = 5 GHz
- Processor B: CPI = 1, clock = 3 GHz
- Probably A, but B is faster… assuming same ISA and same compiler.
- Classic example
  - 800 MHz PentiumIII faster than 1 GHz Pentium4!
  - Same ISA and compiler!
- **Meta-point: danger of partial performance metrics!**

Summarizing Performance

- Arithmetic mean
  - Average execution time
  - Gives more weight to longer-running programs
- Weighted arithmetic mean
  - More important programs can be emphasized
  - But what do we use as weights?
  - Different weight will make different machines look better

Speedup

	Machine A	Machine B
Program 1	5 sec	4 sec
Program 2	3 sec	6 sec

What is the speedup of A compared to B on Program 1?

What is the speedup of A compared to B on Program 2?

What is the average speedup?

What is the speedup of A compared to B on Sum(Program1, Program2) ?

Speedup

	Machine A	Machine B
Program 1	5 sec	4 sec
Program 2	3 sec	6 sec

What is the speedup of A compared to B on Program 1? 0.8x

What is the speedup of A compared to B on Program 2? 2x

What is the average speedup? 1.4x

What is the speedup of A compared to B on Sum(Program1, Program2) ? 1.2x

Mean (Average) Performance Numbers

- **Arithmetic**: $(1/N) * \sum_{P=1..N} \text{Latency}(P)$
  - For units that are proportional to time (e.g., latency)

- You can add latencies (time), but not throughputs (rate/time)
  - Latency(P1+P2,A) = Latency(P1,A) + Latency(P2,A)
  - Throughput(P1+P2,A) != Throughput(P1,A) + Throughput(P2,A)
    - 1 mile @ 30 miles/hour + 1 mile @ 90 miles/hour
    - Average is **not** 60 miles/hour
- **Harmonic**: $N / \sum_{P=1..N} 1/\text{Throughput}(P)$
  - For units that are inversely proportional to time (e.g., throughput)

- **Geometric**: $\sqrt[N]{\prod_{P=1..N} \text{Speedup}(P)}$
  - For unitless quantities (e.g., speedup ratios)

- Suggested reading: Smith on Summarizing performance with a single number… READ IT ☺

Optimizing for performance

- What should you go after first?
  - E.g., what part of your program should you parallelize?

- What is the limit of your performance optimization?

Amdahl's Law (1)

$$\text{Speedup} = \frac{\text{Execution Time without Enhancement}}{\text{Execution Time with Enhancement}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}}$$

What if enhancement does not enhance everything?

$$\text{Speedup} = \frac{\text{Execution Time without using Enhancement at all}}{\text{Execution Time using Enhancement when Possible}}$$

$$\text{Execution Time}_{\text{new}} = \text{Execution Time}_{\text{old}} \times \left(\left(1 - \text{Fraction}_{\text{Enhanced}}\right) + \frac{\text{Fraction}_{\text{Enhanced}}}{\text{Speedup}_{\text{Enhanced}}} \right)$$

$$\text{Overall Speedup} = \frac{1}{\left(\left(1 - \text{Fraction}_{\text{Enhanced}}\right) + \frac{\text{Fraction}_{\text{Enhanced}}}{\text{Speedup}_{\text{Enhanced}}} \right)}$$

Caution: fraction of What?

Amdahl's Law (2)

- Make the Common Case Fast

$$\text{Overall Speedup} = \frac{1}{\left((1 - \text{Fraction}_{\text{Enhanced}}) + \frac{\text{Fraction}_{\text{Enhanced}}}{\text{Speedup}_{\text{Enhanced}}}\right)}$$

$\text{Speedup}_{\text{Enhanced}} = 20$ $\quad \text{Fraction}_{\text{Enhanced}} = 0.1$ VS $\text{Speedup}_{\text{Enhanced}} = 1.2$ $\quad \text{Fraction}_{\text{Enhanced}} = 0.9$

$$\text{Speedup} = \frac{1}{\left((1 - 0.1) + \frac{0.1}{20}\right)} = 1.105$$

$$\text{Speedup} = \frac{1}{\left((1 - 0.9) + \frac{0.9}{1.2}\right)} = 1.176$$

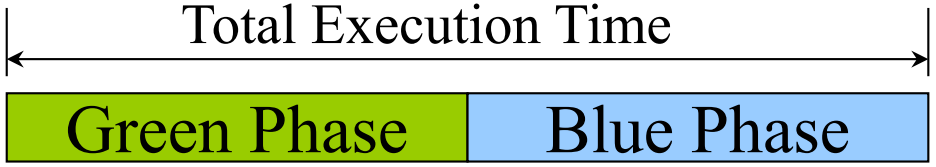
Important: Principle of locality

Approx. 90% of the time spent in 10% of the code

Amdahl's Law (3)

- Diminishing Returns

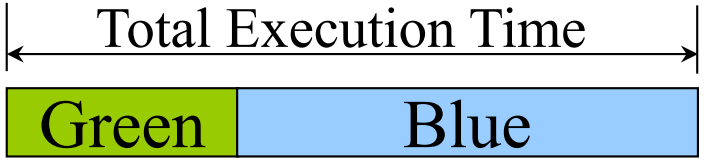
Generation 1



$$\text{Speedup}_{\text{Green}} = 2$$

$$\text{Fraction}_{\text{Green}} = \frac{1}{2}$$

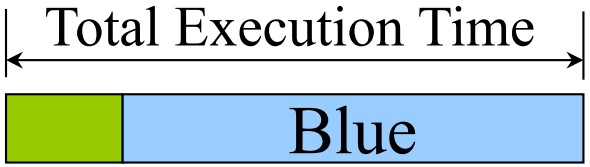
Generation 2 $\text{Speedup}_{\text{Overall}} = 1.33$ over Generation 1



$$\text{Speedup}_{\text{Green}} = 2$$

$$\text{Fraction}_{\text{Green}} = \frac{1}{3}$$

Generation 3 $\text{Speedup}_{\text{Overall}} = 1.2$ over Generation 2



Rules of Thumb

- Make the common case fast
  - driving force behind the RISC philosophy
    - easier for compilers to optimize, simpler decoding
  - Design for actual performance, not peak performance
  - Amdahl's law
- Locality of reference (90/10 rule)
  - programs spend 90% of their time in 10% of the code
  - main principle behind caches (spatial/temporal locality)
- Smaller is faster
  - Why?
  - main principle behind memory hierarchies
    - give illusion of fast, large memory

Performance Trends

	386	486	Pentium	PentiumII	Pentium4	Core2
Year	1985	1989	1993	1998	2001	2006
Technode (nm)	1500	800	350	180	130	65
Transistors (M)	0.3	1.2	3.1	5.5	42	291
Clock (MHz)	**16**	**25**	**66**	**200**	**1500**	3000
Pipe stages	"1"	5	5	10	22	~15
(Peak) IPC	0.4	1	2	3	**3**	**"8"**
(Peak) MIPS	6	25	132	600	4500	24000

- Historically, clock provides 75%+ of performance gains…
  - Achieved via both faster transistors and deeper pipelines
- … that's changing: 1GHz: '99, 2GHz: '01, 3GHz: '02, 4Ghz?
  - Deep pipelining is not power efficient
  - Physical scaling limits are approaching

Producing Wrong Data Without Doing Anything Obviously Wrong!

Todd Mytkowicz Amer Diwan
Department of Computer Science
University of Colorado
Boulder, CO, USA
{mytkowit,diwan}@colorado.edu

Matthias Hauswirth
Faculty of Informatics
University of Lugano
Lugano, CH
Matthias.Hauswirth@unisi.ch

Peter F. Sweeney
IBM Research
Hawthorne, NY, USA
pfs@us.ibm.com

Power/Energy

- Why are they important?

Power/Energy: Increasingly Important

- **Battery life** for mobile devices
  - Laptops, phones, cameras
  - Size too!

- **Tolerable temperature** for devices without active cooling
  - Power means temperature, active cooling means **cost**
  - No room for a fan in a cell phone, no market for a hot cell phone

- **Electric bill** for compute/data centers
  - Pay for power twice: once in, once out (to cool)

- **Environmental concerns**
  - "Computers" account for growing fraction of energy consumption

Btw: Energy & Power

- **Energy**: measured in Joules or Watt-seconds
  - Total amount of energy stored/used
  - Battery life, electric bill, environmental impact
  - Instructions per Joule (car analogy: miles per gallon)
- **Power**: energy per unit time (measured in Watts)
  - Related to "performance" (which is also a "per unit time" metric)
  - Power impacts power supply and cooling requirements (cost)
    - Power-density (Watt/mm^2): important related metric
  - Peak power vs average power
    - E.g., camera, power "spikes" when you actually take a picture
  - Joules per second (car analogy: gallons per hour)
- Two sources:
  - **Dynamic power**: active switching of transistors
  - **Static power**: leakage of transistors even while inactive

Trends in Power

	386	486	Pentium	PentiumII	Pentium4	Core2
Year	1985	1989	1993	1998	2001	2006
Technode (nm)	1500	800	350	180	130	65
Transistors (M)	0.3	1.2	3.1	5.5	42	291
Voltage (V)	**5**	**5**	**3.3**	**2.9**	**1.7**	**1.1**
Clock (MHz)	16	25	66	200	1500	3000
Power (W)	1	5	16	35	**80**	**75**
Peak MIPS	6	25	132	600	4500	24000
MIPS/W	6	5	8	17	56	320

- Supply voltage decreasing over time
- Emphasis on power starting around 2000
  - Resulting in slower frequency increases

Saving Power

- DVFS
- Power savings mode
- Parallelism? Why?
- Other software tricks?
  - Embedded systems
  - Data-centers

- Other ideas?

- Paper reading assignment: Power as first class concern