

Evaluating Computers: Bigger, better, faster, more?

Key Points

- What does it mean for a computer to fast?
- What is latency?
- What is the performance equation?

What do you want in a computer?

- Crysis -- Frame rate
- Responsiveness
- Real time -- deadlines, responsiveness
- Operations/sec -- throughput
- Cost
- Volume
- Weight
- Battery life
- Power/energy consumption
- Storage size
- Reliability
- Latency -- how long does it take?
- Happiness
- Usability
- Flexibility
- Shinyess
- Connectivity -- interesting IO

What do you want in a computer?

- Low latency -- one unit of work in minimum time
 - $1/\text{latency} = \text{responsiveness}$
- High throughput -- maximum work per time
 - High bandwidth (BW)
- Low cost
- Low power -- minimum joules per time
- Low energy -- minimum joules per work
- Reliability -- Mean time to failure (MTTF)
- Derived metrics
 - responsiveness/dollar
 - BW/\$
 - BW/Watt
 - Work/Joule
 - Energy * Latency -- Energy delay product
 - MTTF/\$

Latency

- This is the simplest kind of performance
- How long does it take the computer to perform a task?
 - The task at hand depends on the situation.
- Usually measured in seconds
- Also measured in clock cycles
 - Caution: if you are comparing two different system, you must ensure that the cycle times are the same.

Measuring Latency

- Stop watch!
- System calls
 - `gettimeofday()`
 - `System.currentTimeMillis()`
- Command line
 - `time <command>`

Where latency matters

- Application responsiveness
 - Any time a person is waiting.
 - GUIs
 - Games
 - Internet services (from the users perspective)
- “Real-time” applications
 - Tight constraints enforced by the real world
 - Anti-lock braking systems -- “hard” real time
 - Manufacturing control
 - Multi-media applications -- “soft” real time

Latency and Performance

- By definition:
- $\text{Performance} = 1/\text{Latency}$
- If $\text{Performance}(X) > \text{Performance}(Y)$, X is faster.
- If $\text{Perf}(X)/\text{Perf}(Y) = S$, X is S times faster than Y.
- Equivalently: $\text{Latency}(Y)/\text{Latency}(X) = S$
- When we need to talk about specifically about other kinds of “performance” we must be more specific.

The Performance Equation

- We would like to model how architecture impacts performance (latency)
- This means we need to quantify performance in terms of architectural parameters.
 - Instructions -- this is the basic unit of work for a processor
 - Cycle time -- these two give us a notion of time.
 - Cycles
- Latency equation for simple machines:

$$\text{Latency} = \text{Instructions} * (\text{Cycles / Instruction}) / \text{Freq}$$

The Performance Equation

$$\text{Latency} = \text{Instructions} * \text{Cycles/Instruction} * \text{Seconds/Cycle}$$

- The units work out!
- Cycles/Instruction == CPI
- Seconds/Cycle == 1/hz == 1 / Freq
- Example:
 - 1GHz clock
 - 1 billion instructions
 - CPI = 4
 - What is the latency?

Be careful when comparing systems running the same program

Latency = Instructions * Cycles/Instruction * Seconds/Cycle

*Machine A has the **same** frequency as Machine B; and has a **lower** CPI, but the program runs **slower**. Why? Etc.*

- Different Instruction count?
 - Different ISAs ?
 - Different compilers ?
- Different CPI?
 - underlying machine implementation
 - microarchitecture
- Different cycle time?
 - New process technology
 - Microarchitecture

Computing Average CPI

- Instruction execution time depends on instruction time (we'll get into why this is so later on)

Operation	Integer or Floating Point	Typical Latency in Cycles
+, -, <<, , &	Int	1
/	Int	32
*	Int	2
+ - *	FP	4
/ Sqrt	FP	12
Loads, Stores	Int or FP	3, 8, 30, or 200

- Total CPI depends on the workload's Instruction mix
 - how many of each type of instruction executes
 - What program is running?
 - How was it compiled?

The Compiler's Role

- Compilers affect CPI...
 - Wise instruction selection
 - “Strength reduction”: $x * 2^n \rightarrow x \ll n$
 - Use registers to eliminate loads and stores
 - More compact code $\rightarrow$ less waiting for instructions
- ...and instruction count
 - Common sub-expression elimination
 - Use registers to eliminate loads and stores

Stupid Compiler

```
int i, sum = 0;  
for(i=0;i<10;i++)  
    sum += i;
```

Type	CPI	Static #	dyn #
mem	5	6	42
int	1	3	30
br	1	2	20
Total	2.8	11	92

$$(5 \cdot 42 + 1 \cdot 30 + 1 \cdot 20) / 92 = 2.8$$

```
sw    0($sp), $0 #sum = 0  
sw    4($sp), $0 #i = 0  
loop:  
lw    $1, 4($sp)  
sub    $3, $1, 10  
beq    $3, $0, end  
lw    $2, 0($sp)  
add    $2, $2, $1  
st    0($sp), $2  
addi   $1, $1, 1  
st    4($sp), $1  
b      loop  
end:
```

Smart Compiler

```
int i, sum = 0;  
for(i=0;i<10;i++)  
    sum += i;
```

Type	CPI	Static #	dyn #
mem	5	1	1
int	1	5	32
br	1	2	20
Total	1.01	8	53

$$(5*1 + 1*32 + 1*20)/53 = 1.01$$

```
add    $1, $0, $0 # i  
add    $2, $0, $0 # sum  
loop:  
sub     $3, $1, 10  
beq     $3, $0, end  
add     $2, $2, $1  
addi    $1, $1, 1  
b       loop  
end:  
sw      0($sp), $2
```

Program inputs affect CPI too!

```
int rand[1000] = { random 0s and 1s }  
for (i=0; i<1000; i++)  
    if (rand[i]) sum -= i;  
    else sum *= i;
```

```
int ones[1000] = {1, 1, ...}  
for (i=0; i<1000; i++)  
    if (ones[i]) sum -= i;  
    else sum *= i;
```

- Data-dependent computation
- Data-dependent micro-architectural behavior
 - Processors are faster when the computation is predictable (more later)

Making Meaningful Comparisons

$$\text{Latency} = \text{Instructions} * \text{Cycles/Instruction} * \text{Seconds/Cycle}$$

- Meaningful CPI exists only:
 - For a particular program with a particular compiler
 - with a particular input.
- You MUST consider all 3 to get accurate latency estimations or machine speed comparisons
 - Instruction Set
 - Compiler
 - Implementation of Instruction Set (386 vs Pentium)
 - Processor Freq (600 Mhz vs 1 GHz)
 - Same high level program with same input
- “wall clock” measurements are always comparable.
 - If the workloads (app + inputs) are the same

Limits on Speedup: Amdahl's Law

- “The fundamental theorem of performance optimization”
- Coined by Gene Amdahl
 - one of the designers of the IBM 360
- Optimizations do not (generally) uniformly affect the entire program
 - The more widely applicable a technique is, the more valuable it is
 - Conversely, limited applicability can (drastically) reduce the impact of an optimization.

Always heed Amdahl's Law!!!

It is central to many many optimization problems



Amdahl's Law in Action

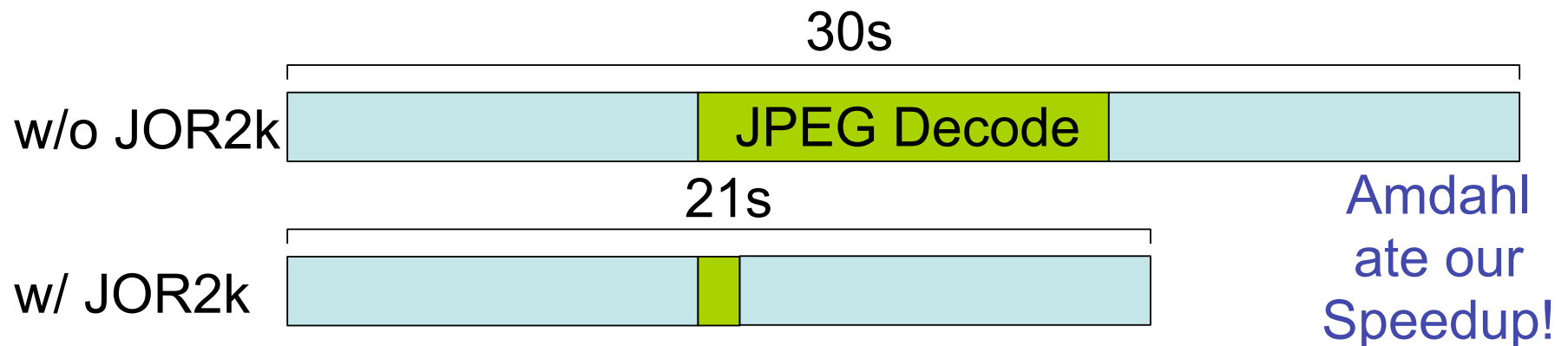
- SuperJPEG-O-Rama2010 ISA extensions **
 - Speeds up JPEG decode by 10x!!!
 - Act now! While Supplies Last!

**

Increases processor cost by 45%

Amdahl's Law in Action

- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of its time doing JPEG decode
- How much does JOR2k help?



Performance: $30/21 = 1.4x$ Speedup $\neq 10x$

Is this worth the 45% increase in cost?

Amdahl's Law

- A fundamental theorem of computer architecture.
- If we can speed up X of the program by S times
- Amdahl's Law gives the total speed up, S_{tot}

$$S_{tot} = \frac{1}{(x/S + (1-x))}$$

Sanity check:

$$x = 1 \Rightarrow S_{tot} = \frac{1}{(1/S + (1-1))} = \frac{1}{1/S} = S$$

Amdahl's Corollary #1

- Maximum possible speedup, S_{max}

$$S = \textit{infinity}$$

$$S_{max} = \underline{1 / (1-x)}$$

Amdahl's Law Example #1

- Protein String Matching Code
 - 200 hours to run on current machine, spends 20% of time doing integer instructions
 - How much faster must you make the integer unit to make the code run 10 hours faster?
 - How much faster must you make the integer unit to make the code run 50 hours faster?

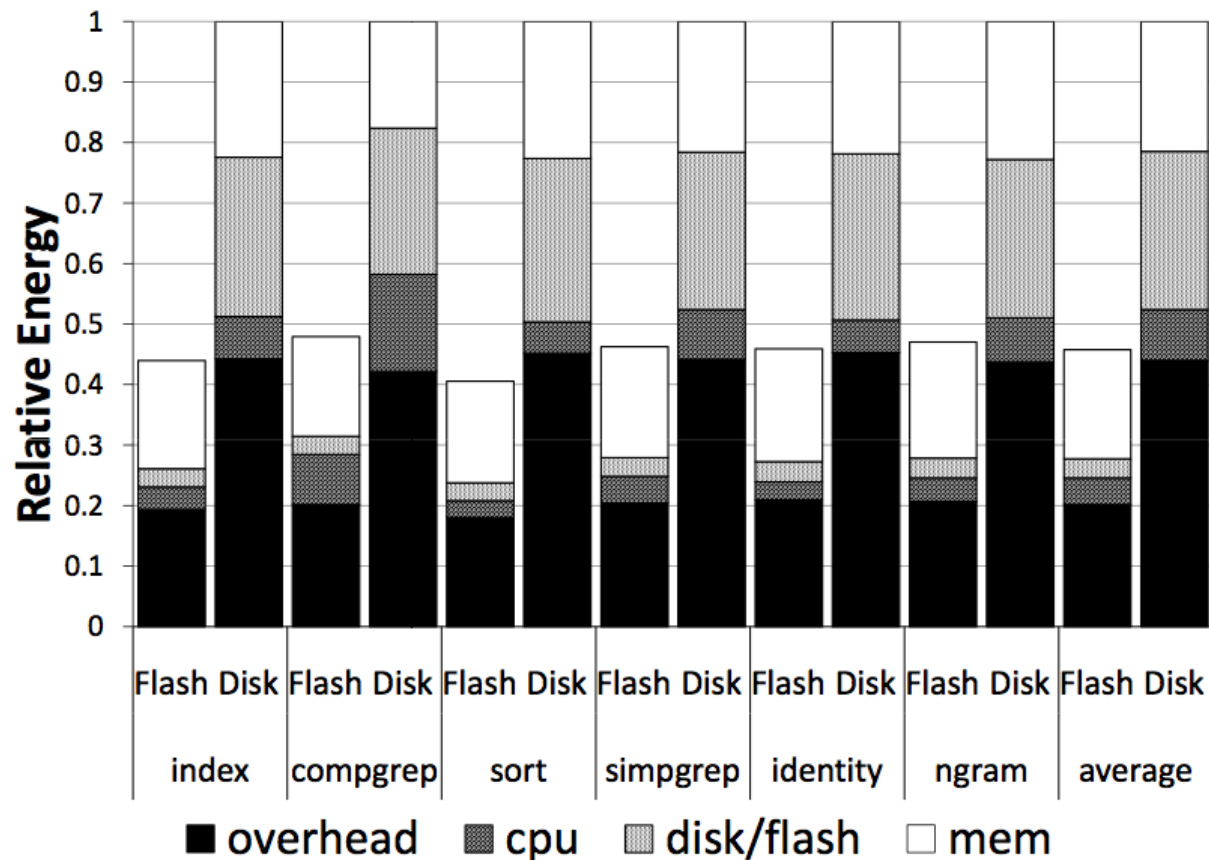
A) 1.1
B) 1.25
C) 1.75
D) 1.33

E) 10.0
F) 50.0
G) 1 million times
H) Other

Amdahl's Law Example #2

- Protein String Matching Code
 - 4 days execution time on current machine
 - 20% of time doing integer instructions
 - 35% percent of time doing I/O
 - Which is the better tradeoff?
 - Compiler optimization that reduces number of integer instructions by 25% (assume each integer inst takes the same amount of time)
 - Hardware optimization that reduces the latency of each IO operations from 6us to 5us.

Amdahl's Law Applies All Over

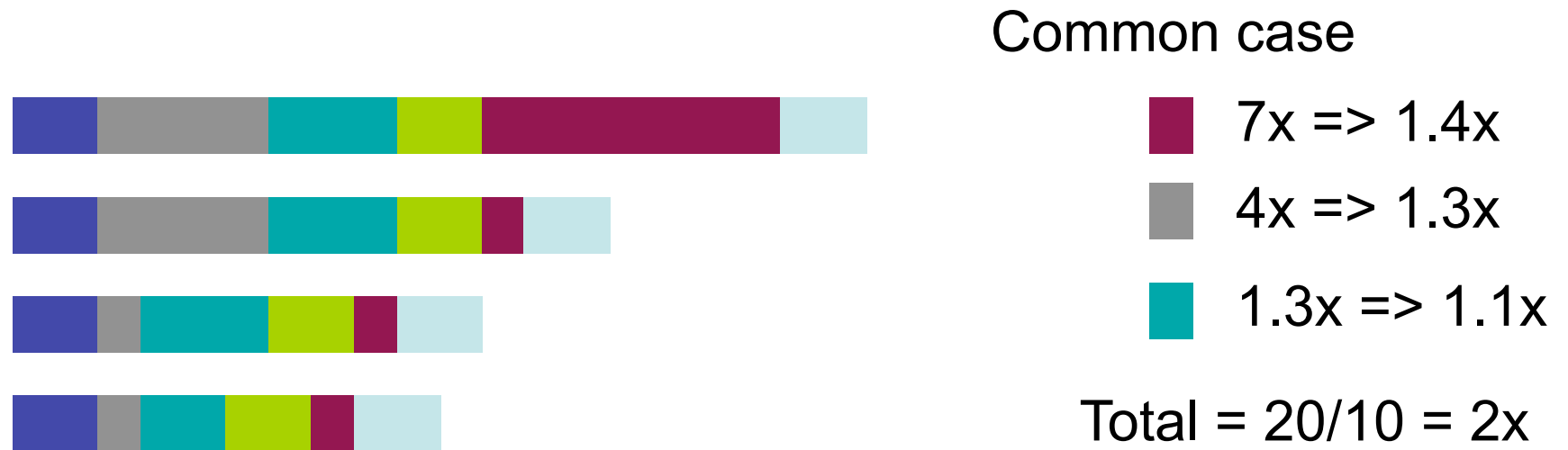


- SSDs use 10x less power than HDs
- But they only save you ~50% overall.

Amdahl's Corollary #2

- Make the common case fast (i.e., x should be large)!
 - Common == “most time consuming” not necessarily “most frequent”
 - The uncommon case doesn't make much difference
 - Be sure of what the common case is
 - The common case changes.
- Repeat...
 - With optimization, the common becomes uncommon and vice versa.

Amdahl's Corollary #2: Example



- In the end, there is no common case!
- Options:
 - Global optimizations (faster clock, better compiler)
 - Find something common to work on (i.e. memory latency)
 - War of attrition
 - Total redesign

Amdahl's Corollary #3

- Benefits of parallel processing
- p processors
- $x\%$ is p -way parallizable
- maximum speedup, S_{par}

$$S_{par} = \frac{1}{(x/p + (1-x))}.$$

x is pretty small for desktop applications, even for $p = 2$

Does Intel's 80-core processor make much sense?

Example #3

- Recent advances in process technology have double the number transistors you can fit on your die.
- Currently, your key customer can use up to 16 processors for 50% of their application.
- You have two choices:
 - Double the number of processors from 2 to 4
 - Stick with 2 processors but add features that will allow the applications to use two cores for 70% of execution.
- Which will you choose?

Bandwidth

- The amount of work (or data) per time
 - MB/s, GB/s -- network BW, disk BW, etc.
 - Frames per second -- Games, video transcoding
- Also called “throughput”

Measuring Bandwidth

- Measure how much work is done
- Measure latency
- Divide

Latency-BW Trade-offs

- Increasing bandwidth can often increase latency for one task.
 - Think of waiting in line for one of 4 bank tellers
 - If the line is empty, your response time is minimized, but throughput is low because utilization is low.
 - If there is always a line, you wait longer (your latency goes up), but there is always work available for tellers.
- Much of computer performance is about scheduling work onto resources
 - Network links.
 - Memory ports.
 - Processors, functional units, etc.
 - IO channels.
 - Increasing contention for these resources generally increases throughput but hurts latency.

Reliability Metrics

- Mean time to failure (MTTF)
 - Average time before a system stops working
 - Very complicated to calculate for complex systems
- Why would a processor fail?
 - Electromigration
 - High-energy particle strikes
 - cracks due to heat/cooling
- It used to be that processors would last longer than their useful life time. This is becoming less true.

Power/Energy Metrics

- Energy == joules
 - You buy electricity in joules.
 - Battery capacity is in joules
 - To minimize operating costs, minimize energy
 - You can also think of this as the amount of work that computer must actually do
- Power == joules/sec
 - Power is how fast your machine uses joules
 - It determines battery life
 - It also determines how much cooling you need. Big systems need 0.3-1 Watt of cooling for every watt of compute.

Power in Processors

- $P = aCV^2f$
 - a = activity factor (what fraction of the xtrs switch every cycles)
 - C = total capacitance (i.e, how many xtrs there are on the chip)
 - V = supply voltage
 - f = clock frequency
- Generally, f is linear in V , so P is roughly f^3
- Architects can improve
 - a -- make the micro architecture more efficient. Less useless xtr switchings
 - C -- smaller chips, with fewer xtrs

Metrics in the wild

- Millions of instructions per second (MIPS)
- Floating point operations per second (FLOPS)
- Giga-(integer)operations per second (GOPS)
- Why are these all bandwidth metric?
 - Peak bandwidth is workload independent, so these metrics describe a hardware capability
 - When you see these, they are generally GNTE (Guaranteed not to exceed) numbers.

More Complex Metrics

- For instance, want low power and low latency
 - $\text{Power} * \text{Latency}$
- More concerned about Power?
 - $\text{Power}^2 * \text{Latency}$
- High bandwidth, low cost?
 - $(\text{MB/s})/\$$
- In general, put the good things in the numerator, the bad things in the denominator.
 - MIPS^2/W

Stationwagon Digression

- IPv6 Internet 2: 272,400 terabit-meters per second
 - 585GB in 30 minutes over 30,000 Km
 - 9.08 Gb/s



- Subaru outback wagon
 - Max load = 408Kg
 - 21Mpg
- MHX2 BT 300 Laptop drive
 - 300GB/Drive
 - 0.135Kg
- 906TB
- Legal speed: 75MPH (33.3 m/s)
- BW = 8.2 Gb/s
- Latency 30k km @ 75 mph = 10 days
- 241,535 terabit-meters per second

What affects Performance

- $\text{Latency} = \text{InstCount} * \text{CPI} * \text{CycleTime}$

	Inst Count	CPI	Cycle time
Program	x		
Compiler	x	(x)	
Inst. Set.	x	x	(x)
Implementation		x	x
Technology			x

Benchmarks: Standards for Performance Measurement

- It's hard to convince manufacturers to run *your* program (unless you're a BIG customer)
- A benchmark is a set of programs that are representative of a class of problems.
- To increase predictability, collections of benchmark applications, called *benchmark suites*, are popular
 - “Easy” to set up
 - Portable
 - Well-understood
 - Stand-alone
 - Standardized conditions
 - These are all things that real software is not.

Classes of benchmarks

- **Microbenchmark** – measure one feature of system
 - e.g. memory accesses or communication speed
- **Kernels** – most compute-intensive part of applications
 - e.g. Linpack and NAS kernel b'marks (for supercomputers)
- Full application:
 - **SpecInt** / **SpecFP** (int and float) (for Unix workstations)
 - Other suites for databases, web servers, graphics,...
 - Demo: spec.org