

Performance, Part 2

Getting a lot done in little time

Performance Recap

- Latency is the time it takes to perform a single task
 - $\text{Performance}(X) = 1 / \text{Latency}(X)$
- Speedup of X over Y:
$$S = \text{Perf}(X) / \text{Perf}(Y) = \text{Latency}(Y) / \text{Latency}(X)$$
- Amdahl's Law limits speedup from optimization
 - Corollary 1: Max speedup bounded by time in optimization
 - Corollary 2: Make the “common case” fast
 - Corollary 3: Parallel speedup limited by time in serial part of program

Latency versus Throughput

e.g. Time versus Bandwidth

Vehicle	Time to Bay Area	Speed	Passengers	(pm/h)
Ferrari	3.1 hours	160 mph	2	320
Greyhound	7.7 hours	65 mph	60	3900

- ° Time to do the task from start to finish
 - "execution time", "latency", "response time"
- ° Tasks per unit time
 - "throughput",

Latency versus Throughput

- Execution Time, or Latency, is measured in *time*.
- Throughput, or Bandwidth, is a *rate*, typically *work/time*.
 - *Incremental amount of work*
(e.g., instructions, bytes, operations) for a given span of time.
 - $\text{Throughput} = \text{Work} / \text{Time}$
 $\therefore \text{Time} = \text{Work} / \text{Throughput} ?$

*My farm can grow 8,760 tomatoes in a year;
How long will it take to grow one tomato?*

Throughput = 8760 tomatoes/year

Work = 1 tomato

Time = 1 tomato / (8760 tomatoes/year) = 1 hour ??

- *In the context of throughput, latency is the time that it takes, from start to finish, for one item to pass through the system.*
 - *UCSD Throughput: 5,000 undergraduates per year*
 - *UCSD Latency: 4.3 years (average)*

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.

The Infamous Bank Line Example

- Or Less Ambiguously, the Fry's Electronics Line

- Two meanings of “**latency typically increases with throughput**”
 - We often think about how we design a system that has been optimized for latency, and then change it to improve its throughput. Assuming the baseline system is optimized, we will often have to sacrifice on latency in order to gain throughput.
 - e.g, If UCSD's focus was latency, we could reduce the enrollment so that we had class sizes of one.
 - If we take an existing system, the unloaded latency (*time through when there are few customers*) is usually less than the loaded latency (*time through when there are more customers*)

E.g., no waiting in line,
less congestion.



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
 - Spilled your soda all over it
- 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 is 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 (fraction of the transistors that switch every cycle)
 - C = total capacitance (i.e, how many transistors 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 transistor switchings
 - C : smaller chips, with fewer transistors

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
 - Latency?



- Subaru outback wagon
 - Max load = 408Kg
- MHX2 BT 300 Laptop drive
 - 300GB/Drive
 - 0.135Kg
- 906TB
- Average 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	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