Metrics
Latency != Bandwidth

Latency is a time. (e.g., seconds, cycles)
- How long for a unit to get through the pipe.

Bandwidth is a rate. (e.g., Gbps, op/s, transactions/s)
- How much you stuff into the pipe per unit time.

<table>
<thead>
<tr>
<th></th>
<th>Latency</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAUI (10G Ethernet)</td>
<td>500 ns</td>
<td>10 Gbps</td>
</tr>
<tr>
<td>Disk</td>
<td>10,000,000 ns</td>
<td>1 Gbps</td>
</tr>
<tr>
<td>DRAM (processor)</td>
<td>30 ns</td>
<td>64 Gbps</td>
</tr>
</tbody>
</table>

UCSD undergrad latency: 4 years
UCSD undergrad bandwidth: .86 students / hour
Low Energy != Low Power

Energy is how much computation costs.

Power is how much we burn per unit time.

Vegas exploits Power/Energy confusion: People spend $ quickly (power) in an effort to appear as if they have a lot of $ (energy.)

Energy: 100 pJ/instruction (45 nm, 1-issue pipeline)

<table>
<thead>
<tr>
<th>Platform</th>
<th>Watts</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>120</td>
</tr>
<tr>
<td>desktop</td>
<td>80</td>
</tr>
<tr>
<td>laptop</td>
<td>30</td>
</tr>
</tbody>
</table>
Energy vs. Delay

Generally, we can burn less energy per op (CV^2) if we are okay with greater delay.

→ Lower V through Voltage+Frequency Scaling
→ Lower C through simpler circuits
  
  single-issue versus 4-way out-of-order pipe
  simple adder versus fast adder

→ Lower C via downsizing gates

How do we normalize between the two?

→ Energy * Delay (Same idea as geomean)
→ Energy * Delay^2
→ Pareto Curves
Pareto Curves

Each point is a design with many parameters projected onto a 2-D plane of two orthogonal metrics.
Speedup

Architect’s measure of improvement.
Speedup > 1.0 is good
Speedup = 1.0 means no change
0 < Speedup < 1.0 means it got slower.

$$\text{Speedup} = \frac{\text{before time}}{\text{after time}}$$
Benchmarks: Comparing Performance

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,...
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 is

– Conversely, limited applicability can (drastically) reduce the impact of an optimization.

Always heed Amdahl’s Law!!!

It is central to many many many optimization problems
How to Summarize Performance

- Arithmetic Mean
  \[ \frac{1}{n} \sum_{i=1}^{n} \text{Time}_i \]

- Weighted Arithmetic Mean
  \[ \sum_{i=1}^{n} \text{Time}_i \times \text{Weight}_i \text{, where the sum of the weights is 1.} \]

- Harmonic Mean
  \[ \frac{n}{\sum_{i=1}^{n} \frac{1}{\text{Rate}_i}} \]

- Geometric Mean
  \[ \sqrt[n]{\prod_{i=1}^{n} \text{ExecutionTimeRatio}_i} \]

Currently in vogue.

Adapted from Brad Calder’s Slides.
Summarizing Performance

• Even the unweighted arithmetic mean implies a weighting
  – the longer the running time the larger the impact

• Geometric means of normalized execution times are consistent no matter which machine is faster
  – ratios of geometric means never change, and always give equal weight to all benchmarks

• Geometric mean does not necessarily predict execution time for any mix of the programs

Adapted from Brad Calder's Slides.
Amdahl’s Law in Action

SuperJPEG-O-Rama2010 in the wild
PictoBench spends 33% of it’s time doing JPEG decode

How much does JOR2k help?

Performance: $\frac{30}{21} \neq 10x$ Speedup

Ammdahl’s Law in Action
Amdahl’s Law

The second fundamental theorem of computer architecture.

If we can speed up $X$ of the program by $S$ times, then Amdahl’s Law gives the total speed up, $S_{tot}$

$$S_{tot} = \frac{1}{\left(\frac{x}{S} + (1-x)\right)}.$$ 

Sanity check:

$x = 1 \implies S_{tot} = \frac{1}{\left(\frac{1}{S} + (1-1)\right)} = S$
Amdahl’s Corollary #1

Maximum possible speedup, $S_{\text{max}}$

$S = \text{infinity}$

$S_{\text{max}} = \frac{1}{1-x}$
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 #3

Benefits of parallel processing

$p$ processors

$x\%$ is $p$-way parallelizable

maximum speedup, $S_{par}$

\[
S_{par} = \frac{1}{\left(\frac{x}{p} + (1-x)\right)}.
\]

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