# 18-447 Lecture 5: Performance and All That (Uniprocessor)

James C. Hoe
Department of ECE
Carnegie Mellon University

# Housekeeping

- Your goal today
  - appreciate the subtleties of measuring/summarizing/comparing performance
  - focus is on sequential execution performance
    - L12: power&energy; L23: parallel performance
- Notices
  - Lab 1, Part A, due this week
  - Lab 1, Part B, due next week
  - HW1, due Monday 2/22
- Readings
  - P&H Ch 1.6~1.9
  - P&H Appendix C for next time

# It's about time

- To the first order, **performance $\propto$ 1 / time**

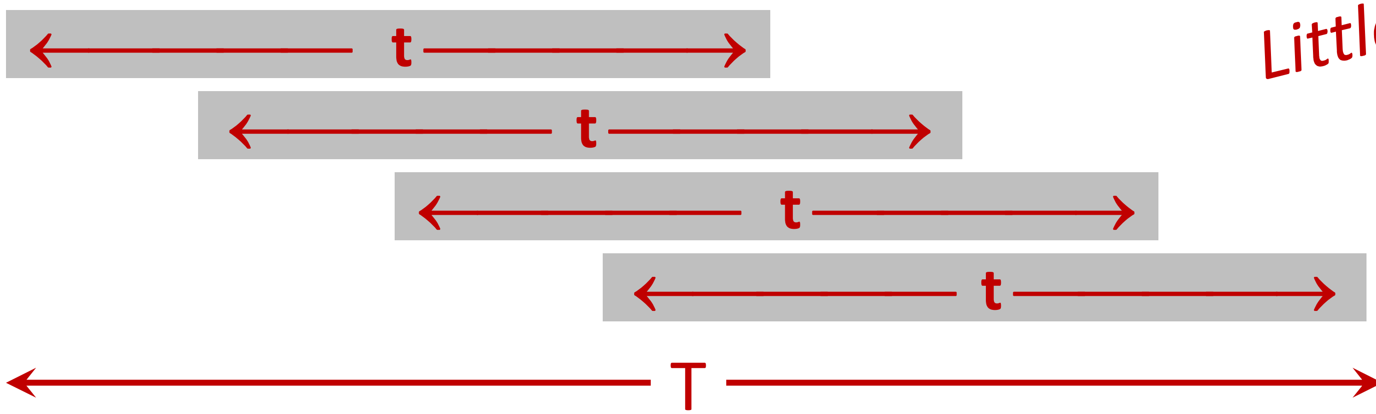
- Two very different kinds of performance!!
  - latency = time between start and finish of a task
  - throughput = number of tasks finished in a given unit of time (a rate measure)

- Either way, shorter the time, higher the performance, but not to be mixed up

# Throughput ≠ 1/Latency

- If it takes t sec to do 1 task, latency=t;
  does throughput=1/t?
- If it takes T sec to do N tasks, throughput=N/T;
  does latency=T/N?
- When there is concurrency, throughput≠1/latency

Little's Law




- Optimizations can tradeoff one for the other

(think school bus vs F1 race car)

# Throughput ≠ Throughput

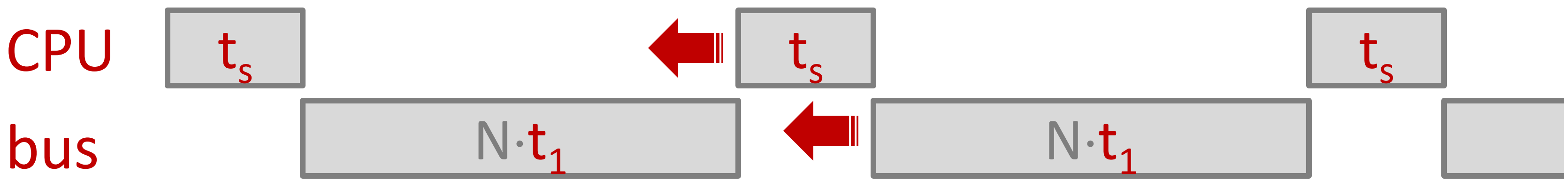
- Throughput becomes a function of N when there is a non-recurring start-up cost (aka overhead)
- For start-up-time=$t_s$ and $\text{throughput}_{raw}=1/t_1$
  - $\text{throughput}_{effective} = N / (t_s + N \cdot t_1)$
  - if $t_s \gg N \cdot t_1$, $\text{throughput}_{effective} \approx N/t_s$
  - if $t_s \ll N \cdot t_1$, $\text{throughput}_{effective} \approx 1/t_1$

  we say $t_s$ is "amortized" in the latter case
- E.g., programmed DMA transfer on a bus
  - bus $\text{throughput}_{raw}$ = 1GByte/sec = 1 Byte / ($10^{-9}$ sec)
  - $10^{-6}$ sec to program a DMA engine
  - $\text{throuhgput}_{effective}$ for transferring 1B/1KB/1MB/1GB?

DMA=Direct Memory Access

# Latency ≠ Latency

- What are you doing during the latency period?
- Latency = hands-on time + hands-off time
- In the DMA example
  - CPU is busy for the $t_s$ to program the DMA engine
  - CPU has to wait $N \cdot t_1$ for DMA to complete
  - CPU could be doing something else during $N \cdot t_1$ to "hide" that latency



# Sounds Like Performance

- The metrics you are most likely to see in microprocessor marketing
  - GHz (billion cycles per second)
  - IPC (instruction per cycle)
  - MIPS (million instructions per second)
- Incomplete and/or misleading
  - GHz and IPC are not (unit work)-per-(unit time)
  - MIPS and IPC are averages (depend on inst mix)
  - GHz, MIPS or IPC can be improved at the expense of each other and actual performance

e.g., 1.4GHz Intel P4 ≈ 1.0GHz Intel P3?

# Iron Law of Processor Performance

- time/program = (inst/program) (cyc/inst) (time/cyc)

  1/IPC 1/MIPS 1/GHz

  note workload dependence

- Contributing factors
  - time/cyc: architecture and implementation
  - cyc/inst: architecture, implementation, instruction mix
  - inst/program: architecture, nature and quality of prgm

- **Note**: cyc/inst is a workload average

  potentially large instantaneous variations due to instruction type and sequence

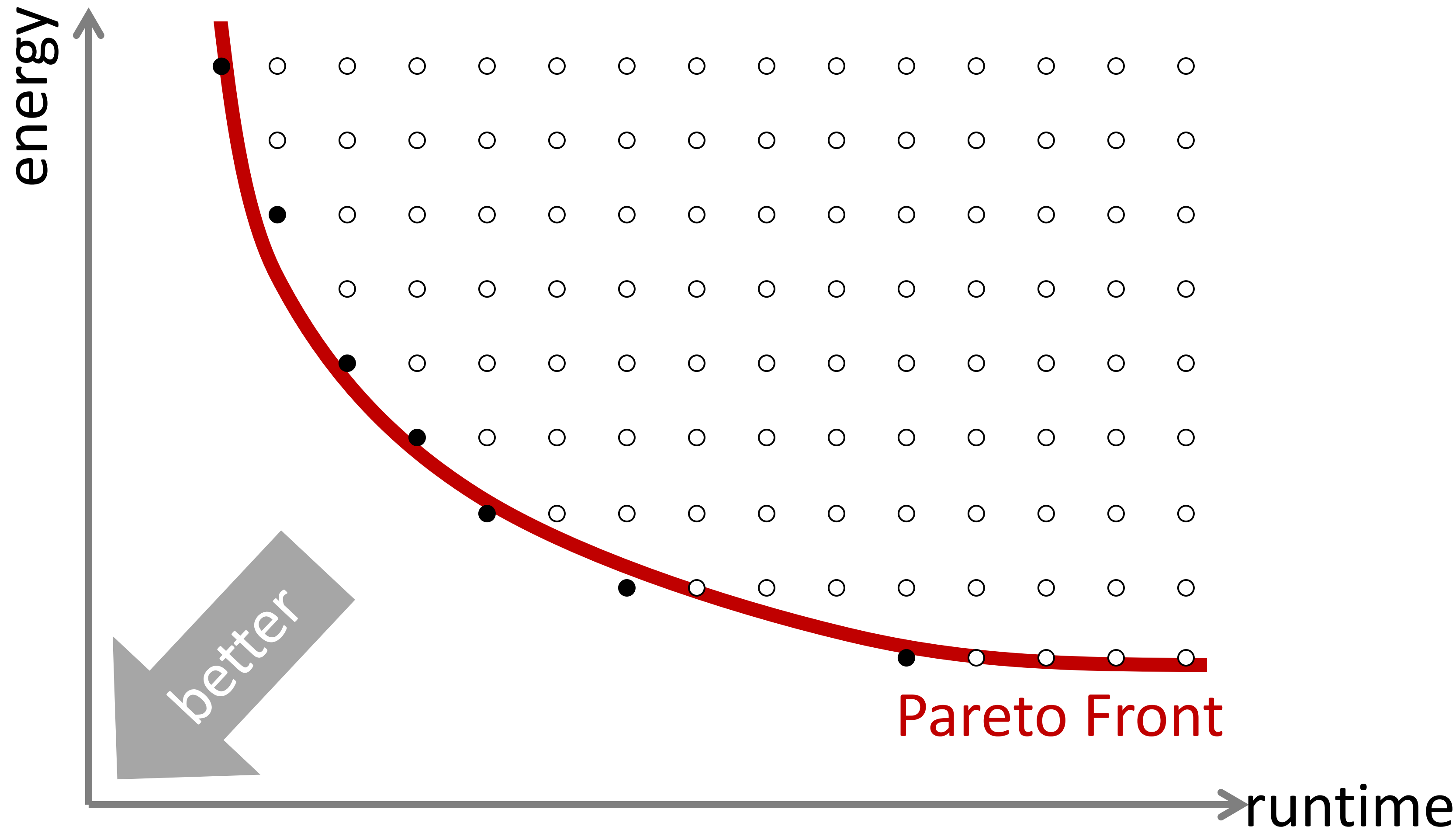
# When it is about more than time

# Tradeoff

- There are other important metric of goodness beside performance: power/energy, cost, risk, social factors . . . ethics . . .
- Cannot optimize individual metrics without considering tradeoff between them
- E.g. runtime vs. energy
  - may be willing to spend more energy per task to run faster
  - conversely, may be willing to run slower for less energy per task
  - but never use more energy to run slower

*"…$5.8 million the value of a statistical life…" FAA*

# Pareto Optimality




All points on front are optimal (can't do better)

How to select between them?

# Composite Metrics

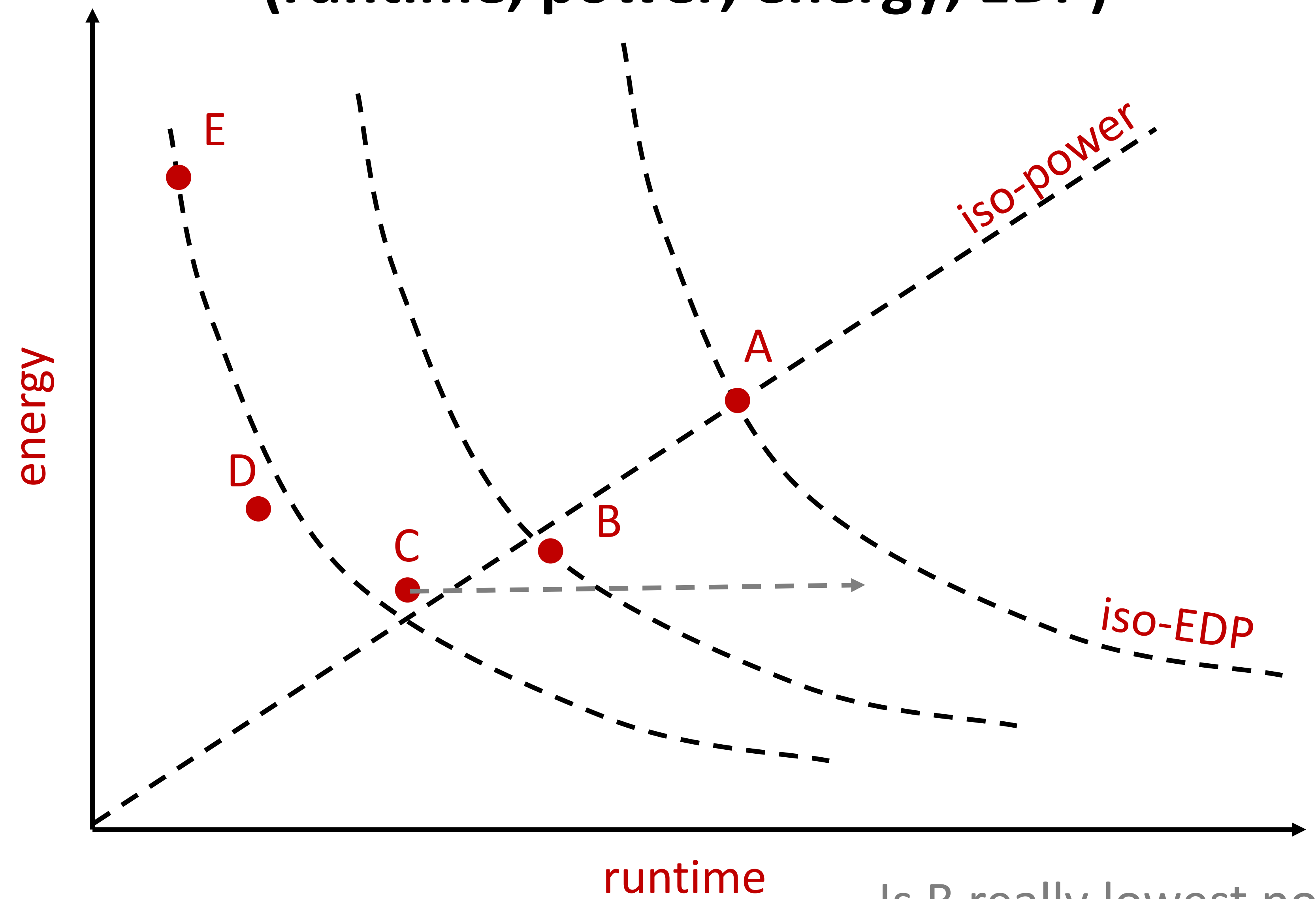
- Define scalar function to reflect desiderata---incorporate dimensions and their relationships
- E.g., energy-delay product
  - can’t cheat by minimizing one ignoring other
  - but is smaller really better?

be wary of relevance to application context

- Floors and ceilings
  - real-life designs more often about good enough than optimal
  - e.g., meet a perf floor under a power(cost)-ceiling

Not all desires reducible to quantifiable terms!!

# Which Design Point is Best?
## (runtime, power, energy, EDP)




Is B really lowest power?

# Comparing and Summarizing Performance

# Relative Performance

- Performance = 1 / Time
  - shorter latency $\Rightarrow$ higher performance
  - higher throughput (job/time) $\Rightarrow$ higher performance

- Pop Quiz

  if X is 50% slower than Y and $Time_X$=1.0s, what is $Time_Y$
  - Case 1: $Time_Y = 0.5s$ since $Time_Y/Time_X=0.5$
  - Case 2: $Time_Y = 0.66666s$ since $Time_X/Time_Y=1.5$

# Architect's Definition of Faster

- "X is n times faster than Y" means

$$n = \text{Performance}_X / \text{Performance}_Y$$
$$= \text{Throughput}_X / \text{Throughput}_Y$$
$$= \text{Time}_Y / \text{Time}_X$$

- "X is m% faster than Y" means

$$1+m/100 = \text{Performance}_X / \text{Performance}_Y$$

- To avoid confusion, stick with definition of "faster"
  - for case 1 say "Y is 100% faster than X"
  - for case 2 say "Y is 50% faster than X"

*According to H&P*

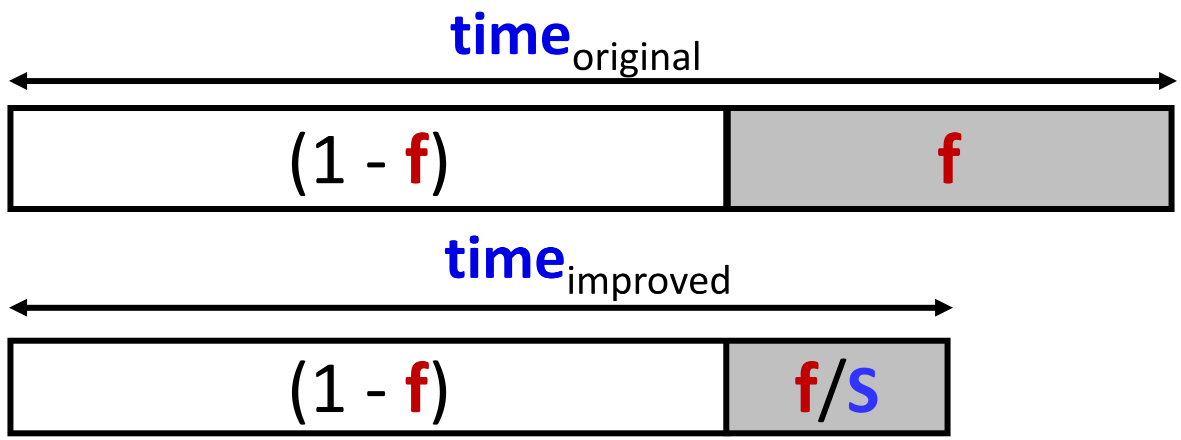
# Architect's Definition of Speedup

- If X is an "enhanced" version of Y, the "speedup" of the enhancement is

$$S = \text{Time}_{\text{without enhancement}} / \text{Time}_{\text{with enhancement}}$$
$$= \text{Time}_Y / \text{Time}_X$$

# Amdahl's Law: a lesson on speedup

- If only a fraction **f** (of time) is speedup by **S**




$$\mathbf{time}_{\text{improved}} = \mathbf{time}_{\text{original}} \cdot \left( (1-\mathbf{f}) + \mathbf{f}/\mathbf{S} \right)$$

$$\mathbf{S}_{\text{effective}} = 1 / \left( (1-\mathbf{f}) + \mathbf{f}/\mathbf{S} \right)$$

  – if **f** is small, **S** doesn't matter
  – even when **f** is large, diminishing return on **S**; eventually "1-**f**" dominates

# Amdahl's Law: a quiz

True or False:

An instruction X is used <u>infrequently</u> (less than 1 in 500 executed instructions) in an embedded workload. Amdahl's Law would say NOT to worry about optimizing the executions of instruction X on a processor designed specifically for that workload.

Hint: what does **f** mean in Amdahl's Law?

# Summarizing Performance

- When comparing two computers X and Y, the relative performance of X and Y depends on program executed
  - X can be m% faster than Y on prog A
  - X can be n% (where m!=n) faster than Y on prog B
  - Y can be k% faster than X on prog C
- Which computer is faster and by how much?
  - depends on which program(s) you care about
  - if multiple programs, also depends their relative importance (frequency or occupancy??)
- Many ways to summarize performance comparisons into a single numerical measure
  - know what the resulting "number" actually mean
  - know when to use which to be meaningful

# Arithmetic Mean

- Suppose workload is applications $A_0, A_1, \ldots, A_{n-1}$
- Arithmetic mean of run time is

$$\frac{1}{n}\sum_{i=0}^{n-1} Time_{A_i}$$

  – comparing AM same as comparing total run-time

caveat: longer running apps have greater contribution than shorter running apps

- If $AM_X/AM_Y=n$ then Y is n times faster than X . . .

**True:** $A_0, A_1, \ldots, A_{n-1}$ run equal number of times always

**False:** some apps run more frequently

Especially bad if most frequent apps also shortest

# Weighted Arithmetic Mean

- Describe relative frequency of apps by weights $w_0, w_1, \ldots, w_{n-1}$
  - $w_i$ = number of $A_i$ executions / total app executions
  - $\sum_{i=0}^{n-1} w_i = 1$
- Weighted AM of the run time = $\sum_{i=0}^{n-1} w_i \cdot Time_{A_i}$
- If $WAM_X / WAM_Y = n$ then Y is n times faster than X on a workload characterized by $w_0, w_1, \ldots, w_{n-1}$
- But $w_i$ isn't always known, so why not "normalize"

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{Time_{A_i\ on\ X}}{Time_{A_i\ on\ Y}} \quad \text{or} \quad \sqrt[n]{\prod_{i=0}^{n-1}\frac{Time_{A_i\ on\ X}}{Time_{A_i\ on\ Y}}}$$

What does it mean though?

# Danger of Normalized Performance

- Suppose
  - $A_0$ takes 1s on X; 10s on Y; 20s on Z
  - $A_1$ takes 1000s on X; 100s on Y; 20s on Z
  - $A_0$+$A_1$ = 1001s on X; 110s on Y; 40s on Z

| | normalized to X | | | normalized to Y | | | normalized to Z | | |
|---|---|---|---|---|---|---|---|---|---|
| | X | Y | Z | X | Y | Z | X | Y | Z |
| $Time_{A0}$ | 1 | 10 | 20 | 0.1 | 1 | 2 | 0.05 | 0.5 | 1 |
| $Time_{A1}$ | 1 | 0.1 | 0.02 | 10 | 1 | 0.2 | 50 | 5 | 1 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| AM of ratio | 1 | 5.05 | 10.01 | 5.05 | 1 | 1.1 | 25.03 | 2.75 | 1 |
| GM of ratio | 1 | 1.0 | 0.63 | 1.0 | 1 | 0.63 | 1.58 | 1.58 | 1 |

[Computer Architecture: A quantitative approach. Hennessy and Patterson]

# Harmonic Mean

- Don't blindly take AM of rates
  - 30mph drive to school (10 miles) and 90mph to return home, roundtrip average speed is not

    (30mph + 90mph)/2
- To compute average rate, expand fully

  average speed = total distance / total time

  = 20 / (10/30 + 10/90) = 45mph
- In case you are not confused,
  - if $A_1$@$IPC_1$, $A_2$@$IPC_2$, ....
  - what is $IPC_{average}$ if $A_1$, $A_2$, ... are equal in **# cyc** vs **# inst** vs **# occurrence**

$$WHM = 1 \Big/ \sum_{i=0}^{n-1} \frac{w_i}{Rate_i}$$

# Standard Benchmarks

- Why standard benchmarks?
  - everyone cares about different applications (different aspects of performance)
  - your application may not be available for the machine you want to study
- E.g. SPEC Benchmarks (www.spec.org)
  - a set of “realistic”, general-purpose, public-domain applications chosen by a multi-industry committee
  - updated every few year to reflect changes in usage and technology
  - a sense of objectivity and predictive power

Everyone knows it is not perfect, but at least everyone plays/cheats by the same rules

# SPEC CPU Benchmark Suites

- CINT2006 (C or C++)

  perlbench (prog lang), bzip2 (compress), gcc (compile),mcf (optimize), gobmk (go), hmmer (gene seq. search), sjeng (chess), libquantum (physics sim.), h264ref (video compress), omnetpp (discrete event sim.), astar (path-finding), xalancbmk (XML)
- CFP2006 (F77/F90 unless otherwise noted)

  bwaves (CFD), gamess (quantum chem), milc (C, QCD), zeusmp (CFD), gromacs (C+Fortran, molecular dyn), cactusADM (C+Fortran, relativity), leslie3d (CFD), namd (C++, molecular dyn), dealII (C++, finite element), soplex (C++, Linear Programming), povray (C++, Ray-trace), calculix (C+Fortran, Finite element), GemsFDTD (E&M), tonto (quantum chem), lbm (C, CFD), wrf (C+Fortran, weather), sphinx3 (C, speech recog)
- Reports GM of performance normalized to a 1997-era 296MHz Sun UltraSparc II

(http://www.spec.org/cpu2006)

# Performance Recap

- There is no one-size-fits-all methodology
  - be sure you understand what you want to measure
  - be sure you understand what you measured
  - be sure what you report is accurate and representative
  - be ready to come clean with raw data
- No one believes your numbers anyway
  - explain what effect you are trying to measure
  - explain what and how you actually measured
  - explain how performance is summarized and represented

When it matters, people will check for themselves

# Most important is to be truthful

*We, the members of the IEEE, in recognition of the importance of our technologies . . . do hereby commit ourselves to the highest ethical and professional conduct and agree:*

> *7. to be honest and realistic in stating claims or estimates based on available data;*

--- Paragraph 7.8 IEEE Code of Ethics, IEEE Policies

# Yeh, et al., "Parachute use to prevent death and major trauma when jumping from aircraft: randomized controlled trial," BMJ 2018.

- *Results: Parachute use did not significantly reduce death or major injury (0% for parachute v 0% for control; P>0.9). This finding was consistent across multiple subgroups. Compared with individuals screened but not enrolled, participants included in the study were on aircraft at significantly lower altitude (mean of 0.6 m for participants v mean of 9146 m for non-participants; P<0.001) and lower velocity (mean of 0 km/h v mean of 800 km/h; P<0.001).*
- https://www.bmj.com/content/363/bmj.k5094