18-447 Lecture 5: Performance (Sequential)

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

◆ Your goal today
  - appreciate the subtleties of measuring/summarizing/comparing performance
  - today’s focus on sequential execution performance
    • L12: power and energy; L23: parallel performance

◆ Notices
  - Lab 1, Part A, due this week
  - Lab 1, Part B, due next week
  - HW1, due next Wed

◆ Readings
  - P&H Ch 1.6, 1.7 and 1.9
  - optional (in supplemental handout on Blackboard)
    • 1991 RISC vs CISC paper
  - P&H Appendix D for next time
It’s about time

Common Notions of “Performance”

- To the first order, performance $\propto \frac{1}{\text{time}}$

- 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)
  - not to be mixed up

- Either way, shorter the time, higher the performance, but . . .
Throughput ≠ 1/Latency

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

- 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 throughput$_{raw}=1/t_1$
  - throughput$_{effective}=N / (t_s + N\cdot t_1)$
  - if $t_s >> N\cdot t_1$, throughput$_{effective} \approx N/t_s$
  - if $t_s << N\cdot t_1$, throughput$_{effective} \approx 1/t_1$
  - we say $t_s$ is “amortized” in the latter case
- E.g., programmed DMA transfer on a bus
  - $10^{-6}$ sec to program a DMA engine
  - bus throughput$_{raw}=1\,\text{GByte/sec} = 1\,\text{Byte} / (10^{-9} \, \text{sec})$
  - throughput$_{effective}$ when 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

When it is about more than time

- There are other important metric of goodness beside performance: power/energy, cost, risk, social factors...
- 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
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
  - smaller the better
  - can’t cheat by minimizing one ignoring other
  - not required to have physical meaning
- Floors and ceilings
  - real-life designs more often about good enough than optimal
  - e.g., meet a perf floor under a power(cost)-ceiling
Which is Design Point is Best?
(runtime, power, energy, EDP)

Is B really lowest power?

Iron Law on Processor Performance
Pseudo Performance

- The metrics you are most likely to see in marketing
  - GHz (cycles per second)
  - IPC (instruction per cycle)
  - MIPS (million instruction per second)

- “Sounds” like performance but incomplete and misleading
  - MIPS and IPC are averages (depend on instruction 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 Performance

- wall clock time = \( \frac{\text{inst}}{\text{program}} \) \( \frac{\text{cyc}}{\text{inst}} \) \( \frac{\text{time}}{\text{cyc}} \)

1/IPC 1/MIPS 1/GHz

Note workload dependence

- Contributing factors
  - \( \frac{\text{time}}{\text{cyc}} \) affected by architecture+implementation
  - \( \frac{\text{cyc}}{\text{inst}} \) affected by architecture+implementation+workload
  - \( \frac{\text{inst}}{\text{program}} \) affected by architecture+workload

- **Note**: \( \frac{\text{cyc}}{\text{inst}} \) is a workload average
  potentially large instantaneous variations due to instruction type and sequence
“RISC Factor” Study  
[Bhandarkar and Clark, 1991]

- Compare VAX and MIPS ISAs on comparable implementation and technology bases

<table>
<thead>
<tr>
<th>benchmark</th>
<th>instruc. ratio</th>
<th>MIPS CPI ratio</th>
<th>VAX CPI ratio</th>
<th>RISC factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>spice2g6</td>
<td>2.48</td>
<td>1.80</td>
<td>8.02</td>
<td>4.44</td>
</tr>
<tr>
<td>matrix300</td>
<td>2.37</td>
<td>3.06</td>
<td>13.81</td>
<td>4.51</td>
</tr>
<tr>
<td>nasa7</td>
<td>2.10</td>
<td>3.01</td>
<td>14.95</td>
<td>4.97</td>
</tr>
<tr>
<td>fpppp</td>
<td>3.88</td>
<td>1.45</td>
<td>15.16</td>
<td>10.45</td>
</tr>
<tr>
<td>tomcatv</td>
<td>2.86</td>
<td>2.13</td>
<td>17.45</td>
<td>8.18</td>
</tr>
<tr>
<td>doduc</td>
<td>2.65</td>
<td>1.67</td>
<td>13.16</td>
<td>7.85</td>
</tr>
<tr>
<td>espresso</td>
<td>1.70</td>
<td>1.06</td>
<td>5.40</td>
<td>5.09</td>
</tr>
<tr>
<td>equitt</td>
<td>1.08</td>
<td>1.25</td>
<td>4.38</td>
<td>3.51</td>
</tr>
<tr>
<td>li</td>
<td>1.62</td>
<td>1.10</td>
<td>6.53</td>
<td>5.97</td>
</tr>
<tr>
<td>geo. mean</td>
<td>2.17</td>
<td>1.71</td>
<td>9.87</td>
<td>5.77</td>
</tr>
</tbody>
</table>

Cycle Time: VAX 4000/300=28ns; MIPS M/2000=40ns

More than HW matters

- Algorithm has first-order effect on performance through inst/program, e.g., discrete Fourier transform
  - 2N^3 FP operations by matrix-multiplication
  - 5N\log_2(N) FP operations by fast algorithms
    
    If N=1024, 2N^3\approx2\times10^9 vs. 5N\log_2(N)\approx5\times10^4

- More abstract programming languages can result in higher inst/program

- Quality of compiler optimizations affect both inst/program and cyc/inst, e.g.,
  - strength-reduction
  - instruction scheduling
Pop Quiz: Pseudo-FLOPS

- Scientific computing community often use pseudo FLOPS as performance metric
  - nominal # of floating point operations
    - program runtime
  - e.g. FFT of size N has nominally $5N \log_2(N)$ FP operations

- Is this a good, fair metric to compare HW+language+compiler+algorithm combinations?
  - not all FFT algorithms have the same FP OP count
  - not all FP OPs are equal (FADD vs FMULT vs FDIV)

Ans: yes, pseudo FLOPS is proportional to 1/time as long as you are computing the same problem

Comparing and Summarizing Performance
Relative Performance

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

- Pop Quiz
  if X is 50% slower than Y and \(\text{Time}_X = 1.0\)s, what is \(\text{Time}_Y\)?
  - Case 1: \(\text{Time}_Y = 0.5\)s since \(\text{Time}_Y / \text{Time}_X = 0.5\)
  - Case 2: \(\text{Time}_Y = 0.66666\)s since \(\text{Time}_X / \text{Time}_Y = 1.5\)

Relative Performance

- “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”
Speedup

- If $X$ is an “enhanced” version of $Y$, the “speedup” of the enhancement is

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

Amdahl’s Law on Speedup

- Suppose an enhancement speeds up a fraction $f$ of a task by a factor of $S_f$

\[
\begin{align*}
\text{time}_{\text{old}} & \quad (1 - f) \quad f \\
\text{time}_{\text{new}} & \quad (1 - f) \quad f/S_f
\end{align*}
\]

$$\text{time}_{\text{new}} = \text{time}_{\text{old}} \cdot (1-f) + \frac{f}{S_f}$$

$$S_{\text{overall}} = \frac{1}{(1-f) + \frac{f}{S_f}}$$

- TAKE AWAY: optimize the common case!
  - $S_{\text{overall}}$ can never exceed $1/(1-f)$
  - $f$ should be a “common” case that dominates runtime (don’t confuse with the “frequent” case)
  - the uncommon becomes more common after $f$ improves
Summarizing Performance

- When comparing two computers \( X \) and \( Y \), the relative performance of \( X \) and \( Y \) depends strongly on what \( X \) and \( Y \) are asked to do
  - \( X \) may be \( m\% \) faster than \( Y \) on application \( A \)
  - \( X \) may be \( n\% \) (where \( m\% \neq n\% \)) faster than \( Y \) on application \( B \)
  - \( Y \) may be \( k\% \) faster than \( X \) on application \( C \)
- Which computer is faster and by how much?
  - depends on which application(s) you care about
  - if you care about several applications, then it also depends their relative importance
- Many ways to summarize performance comparison into a single quantitative measure
  - some may even be meaningful for exactly your purpose
  - but you have to know when to do what
  - when in doubt, present the complete story

Arithmetic Mean

- Suppose you workload is applications \( A_0, A_1, \ldots, A_{n-1} \)
- Arithmetic mean of the application run time is
  \[ \frac{1}{n} \sum_{i=0}^{n-1} Time_{A_i} \]
- comparing AM is the same as comparing total run-time
- caveat: longer applications have greater contribution than shorter applications
- If \( AM_Y / AM_X = n \) then \( Y \) is \( n \) times faster than \( X \ldots \)
  True: \( A_0, A_1, \ldots, A_{n-1} \) are run equal number of times always
  False: if some applications are run much more frequently than others (especially problematic if the most frequent applications are also much shorter than the rest)
Weighted Arithmetic Mean

- Introduce weighting factors, \( w_0, w_1, \ldots, w_{n-1} \) where \( 1 = \sum_{i=0}^{n-1} w_i \)
- \( w_i \) is the number of times \( A_i \) runs relative to total number of times any program in the workload is run
- Weighted arithmetic mean of the run time is
  \[
  \sum_{i=0}^{n-1} w_i \cdot \text{Time}_{A_i}
  \]
- If \( \text{WAM}_X / \text{WAM}_Y = n \) 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 fixed and isn’t easy to come by, how about

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

Yes, you get a number at the end, but what does it mean?

Danger of Normalized Performance

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

<table>
<thead>
<tr>
<th></th>
<th>normalized to X</th>
<th>normalized to Y</th>
<th>normalized to Z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( X )</td>
<td>( Y )</td>
<td>( Z )</td>
</tr>
<tr>
<td>Time(_{A0})</td>
<td>1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Time(_{A1})</td>
<td>1</td>
<td>0.1</td>
<td>0.02</td>
</tr>
<tr>
<td>AM of ratio</td>
<td>1</td>
<td>5.05</td>
<td>10.01</td>
</tr>
<tr>
<td>GM of ratio</td>
<td>1</td>
<td>1.0</td>
<td>0.63</td>
</tr>
</tbody>
</table>

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

- Don’t take arithmetic mean of rates (e.g. throughput)
  - e.g. 30 mph for first 10 miles, 90 mph for next 10 miles, the average speed is not \((30 + 90)/2 = 60\) mph!
- To compute average rate
  1. expand fully
     \[
     \text{average speed} = \frac{\text{total distance}}{\text{total time}}
     \]
     \[
     = \frac{20}{(10/30 + 10/90)} = 45 \text{ mph}
     \]
  2. harmonic mean
     \[
     HM = n \left/ \sum_{i=0}^{n-1} \frac{1}{Rate_i} \right.
     \]
     \[
     WHM = \frac{1}{\sum_{i=0}^{n-1} \frac{W_i}{Rate_i}}
     \]

HM is just a short-cut for doing the fully expanded calculation to average rates

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)
  - Standard Performance Evaluation Corporation
  - 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
(http://www.spec.org/cpu2006)

- **CINT2006** (C unless otherwise noted)
  perlbench (prog lang), bzip2 (compress), gcc (compile), mcf (optimize), gobmk (go), hmmer (gene seq. search), sjeng (chess), libquantum (physics sim.), h264ref (video compress), omnetpp (C++, discrete event sim.), astar (C++, path-finding), xalancbmk (C++, 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), GemsFDTS (E&M), tonto (quantum chem), lbm (C, CFD), wrf (C+Fortran, weather), sphinx3 (C, speech recog)
- Reports geometric mean of normalized performance relative to a 1997-era 296MHz Sun UltraSparc II

---

**Performance Summary**

- 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

If it really matters, people will want to 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