Measurement & Performance

Todd C. Mowry
CS 347

Jan 15, 1998

Topics:

• Timers
• Performance measures
• Relating performance measures
  – system performance measures
  – latency and throughput
  – Amdahl’s law
The Nature of Time

real (i.e. wall clock) time

= **User Time**: time spent executing instructions in the user process

= **System Time**: time spent executing instructions in the *kernel* on behalf of the user process

= all other time (either idle or else executing instructions unrelated to the user process)

real (wall clock) time

Unless otherwise specified, “time” often refers to “user time”.
Anatomy of a Timer

A counter value (T) is updated upon discrete ticks

- a tick occurs once every \( \Delta \) time units
- upon a tick, the counter value is incremented by \( \Delta \) time units

Some Terminology:

- timer \textit{period} = \( \Delta \) seconds / tick
- timer \textit{resolution} = 1/\( \Delta \) ticks / second
Using Timers

Estimating elapsed time:
- based on discrete timer values before ($T_s$) and after ($T_f$) the event

\[ T_{observed} = T_f - T_s \]

How close is $T_{observed}$ to $T_{actual}$?
Timer Error: Example #1

$T_{\text{actual}}$: $\sim 2 \Delta$

$T_{\text{observed}}$: $\Delta$

Absolute measurement error: $\sim \Delta$

Relative measurement error: $\sim \Delta / 2\Delta = \sim 50\%$
Timer Error: Example #2

\[ T_{\text{actual}}: \varepsilon \ (\sim \text{zero}) \]
\[ T_{\text{observed}}: \Delta \]

Absolute measurement error: \( \sim \Delta \)
Relative measurement error: \( \sim \frac{\Delta}{\varepsilon} = \sim \text{infinite} \)
Timer Error: Example #3

\[ T_{\text{actual}}: X \]
\[ T_{\text{observed}}: 0 \]

Absolute measurement error: \( X \)
Relative measurement error: \( X / X = 100\% \)
Absolute measurement error: +/- $\Delta$

Key point: need a large number of ticks to hide error

- can compute $T_{\text{threshold}}$ as a function of $\Delta$ and $E$
- $T_{\text{threshold}} = \text{minimum observed time to guarantee relative error bound}$
- $E = \text{maximum acceptable relative measurement error}$
Homework 1 Timer Package

Unix interval countdown timer

- *decrements* timer value by $\Delta$ every $\Delta$ seconds
- `setitimer()`: initialize timer value
- `getitimer()`: sample timer value
- measures user time

“etime” package:

- based on Unix interval timers
- `set_etime()`: initializes timer
- `get_etime()`: returns elapsed time in seconds since last call to `set_etime()`
Performance expressed as a time

Absolute time measures
- difference between start and finish of an operation
- synonyms: running time, elapsed time, response time, latency, completion time, execution time
- most straightforward performance measure

Relative (normalized) time measures
- running time normalized to some reference time
- (e.g. time/reference time)

Guiding principle: Choose performance measures that track running time.
Performance expressed as a rate

Rates are performance measures expressed in units of work per unit time.

Examples:

- millions of instructions / sec (MIPS)
- millions of floating point instructions / sec (MFLOPS)
- millions of bytes / sec (MBytes/sec)
- millions of bits / sec (Mbits/sec)
- images / sec
- samples / sec
- transactions / sec (TPS)
Key idea: Report rates that track execution time.

Example: Suppose we are measuring a program that convolves a stream of images from a video camera.

Bad performance measure: MFLOPS
  • number of floating point operations depends on the particular convolution algorithm: \( n^2 \) matrix-vector product vs \( n \log n \) fast Fourier transform. An FFT with a bad MFLOPS rate may run faster than a matrix-vector product with a good MFLOPS rate.

Good performance measure: images/sec
  • a program that runs faster will convolve more images per second.
Performance expressed as a rate (cont)

Fallacy: Peak rates track running time.

Example: the i860 is advertised as having a peak rate of 80 MFLOPS (40 MHz with 2 flops per cycle).

However, the measured performance of some compiled linear algebra kernels (icc -O2) tells a different story:

<table>
<thead>
<tr>
<th>Kernel</th>
<th>1d fft</th>
<th>sasum</th>
<th>saxpy</th>
<th>sdot</th>
<th>sgemm</th>
<th>sgemv</th>
<th>spvma</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFLOPS</td>
<td>8.5</td>
<td>3.2</td>
<td>6.1</td>
<td>10.3</td>
<td>6.2</td>
<td>15.0</td>
<td>8.1</td>
</tr>
<tr>
<td>%peak</td>
<td>11%</td>
<td>4%</td>
<td>7%</td>
<td>13%</td>
<td>8%</td>
<td>19%</td>
<td>10%</td>
</tr>
</tbody>
</table>
Relating time to system measures

Suppose that for some program we have:

- \( T \) seconds running time (the ultimate performance measure)
- \( C \) clock ticks, \( I \) instructions, \( P \) seconds/tick (performance measures of interest to the system designer)

\[
T \text{ secs} = C \text{ ticks} \times P \text{ secs/tick} \\
= (I \text{ inst}/I \text{ inst}) \times C \text{ ticks} \times P \text{ secs/tick} \\
T \text{ secs} = I \text{ inst} \times (C \text{ ticks}/I \text{ inst}) \times P \text{ secs/tick}
\]
Pipeline latency and throughput

Latency (L): time to process an individual image.

Throughput (R): images processed per unit time

One image can be processed by the system at any point in time
Video system performance

L = 3 secs/image.
R = 1/L = 1/3 images/sec.
T = L + (N-1)1/R
= 3N
Pipelining the video system

One image can be in each stage at any point in time.

$L_i = \text{latency of stage } i$

$R_i = \text{throughput of stage } i$

$L = L_1 + L_2 + L_3$

$R = \min(R_1, R_2, R_3)$
Suppose:

$L_1 = L_2 = L_3 = 1$

Then:

$L = 3$ secs/image.

$R = 1$ image/sec.

$T = L + (N-1)/R = N + 2$
Relating time to latency and throughput

In general:

• \( T = L + (N-1)/R \)

The impact of latency and throughput on running time depends on \( N \):

• \((N = 1) \implies (T = L)\)
• \((N >> 1) \implies (T = N-1/R)\)

To maximize throughput, we should try to maximize the minimum throughput over all stages (i.e., we strive for all stages to have equal throughput).
Amdahl’s law

You plan to visit a friend in Normandy France and must decide whether it is worth it to take the Concorde SST ($3,100) or a 747 ($1,021) from NY to Paris, assuming it will take 4 hours Pgh to NY and 4 hours Paris to Normandy.

<table>
<thead>
<tr>
<th></th>
<th>time NY-&gt;Paris</th>
<th>total trip time</th>
<th>speedup over 747</th>
</tr>
</thead>
<tbody>
<tr>
<td>747</td>
<td>8.5 hours</td>
<td>16.5 hours</td>
<td>1</td>
</tr>
<tr>
<td>SST</td>
<td>3.75 hours</td>
<td>11.75 hours</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Taking the SST (which is 2.2 times faster) speeds up the overall trip by only a factor of 1.4!
Amdahl’s law (cont)

Old program (unenhanced)

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
</table>

Old time: $T = T_1 + T_2$

New program (enhanced)

<table>
<thead>
<tr>
<th>$T_1'$</th>
<th>$T_2'$</th>
</tr>
</thead>
</table>

$T_1' = T_1$  $T_2' \leq T_2$

New time: $T' = T_1' + T_2'$

Speedup: $S_{\text{overall}} = T / T'$

$T_1 = \text{time that can NOT be enhanced.}$

$T_2 = \text{time that can be enhanced.}$

$T_2' = \text{time after the enhancement.}$
Amdahl’s law (cont)

Two key parameters:

\[ F_{\text{enhanced}} = \frac{T_2}{T} \quad \text{(fraction of original time that can be improved)} \]
\[ S_{\text{enhanced}} = \frac{T_2}{T'} \quad \text{(speedup of enhanced part)} \]

\[ T' = T_1' + T_2' = T_1 + T_2' = T(1-F_{\text{enhanced}}) + T_2' \]
\[ = T(1-F_{\text{enhanced}}) + \left(\frac{T_2}{S_{\text{enhanced}}}\right) \quad \text{[by def of } S_{\text{enhanced}}\text{]} \]
\[ = T(1-F_{\text{enhanced}}) + T(F_{\text{enhanced}}/S_{\text{enhanced}}) \quad \text{[by def of } F_{\text{enhanced}}\text{]} \]
\[ = T((1-F_{\text{enhanced}}) + F_{\text{enhanced}}/S_{\text{enhanced}}) \]

Amdahl’s Law:

\[ S_{\text{overall}} = \frac{T}{T'} = 1/((1-F_{\text{enhanced}}) + F_{\text{enhanced}}/S_{\text{enhanced}}) \]

Key idea: Amdahl’s law quantifies the general notion of diminishing returns. It applies to any activity, not just computer programs.
Trip example: Suppose that for the New York to Paris leg, we now consider the possibility of taking a rocket ship (15 minutes) or a handy rip in the fabric of space-time (0 minutes):

<table>
<thead>
<tr>
<th></th>
<th>time NY-&gt;Paris</th>
<th>total trip time</th>
<th>speedup over 747</th>
</tr>
</thead>
<tbody>
<tr>
<td>747</td>
<td>8.5 hours</td>
<td>16.5 hours</td>
<td>1</td>
</tr>
<tr>
<td>SST</td>
<td>3.75 hours</td>
<td>11.75 hours</td>
<td>1.4</td>
</tr>
<tr>
<td>rocket</td>
<td>0.25 hours</td>
<td>8.25 hours</td>
<td>2.0</td>
</tr>
<tr>
<td>rip</td>
<td>0.0 hours</td>
<td>8 hours</td>
<td>2.1</td>
</tr>
</tbody>
</table>
Useful corollary to Amdahl’s law:

\[ 1 \leq S_{\text{overall}} \leq \frac{1}{(1 - F_{\text{enhanced}})} \]

<table>
<thead>
<tr>
<th>( F_{\text{enhanced}} )</th>
<th>( \text{Max } S_{\text{overall}} )</th>
<th>( F_{\text{enhanced}} )</th>
<th>( \text{Max } S_{\text{overall}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1</td>
<td>0.9375</td>
<td>16</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.96875</td>
<td>32</td>
</tr>
<tr>
<td>0.75</td>
<td>4</td>
<td>0.984375</td>
<td>64</td>
</tr>
<tr>
<td>0.875</td>
<td>8</td>
<td>0.9921875</td>
<td>128</td>
</tr>
</tbody>
</table>

Moral: It is hard to speed up a program.

Moral++: It is easy to make premature optimizations.