Performance

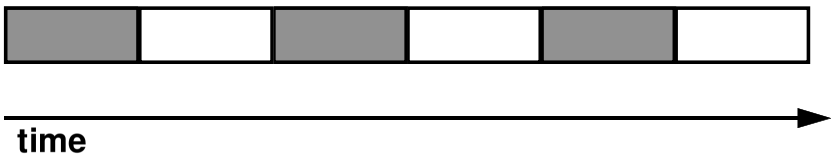
CS 347

Jan. 30, 1996

Topics

- **Measuring programs**
- **Performance measures**
- **Relating performance measures**
  - system perfomance measures
  - latency and throughput
  - Amdahl's law

Measuring programs

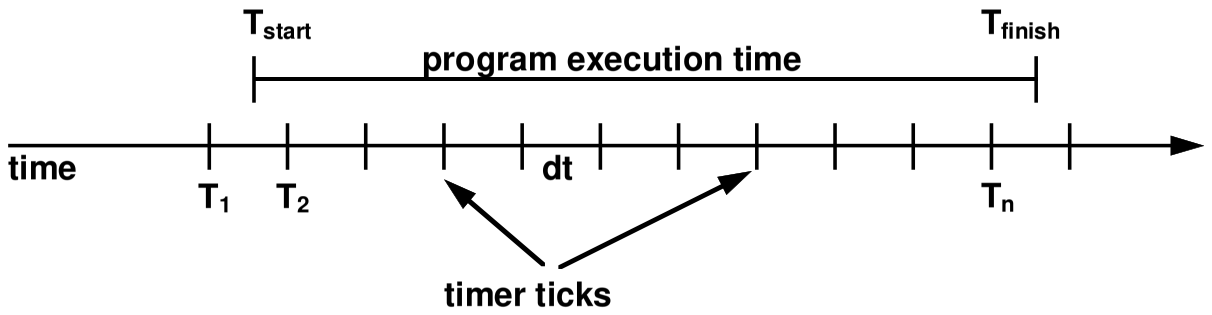


= user time (time executing instructing instructions in the user's process)

+ = real (wall clock) time

We will use the word "time" to refer to user time.

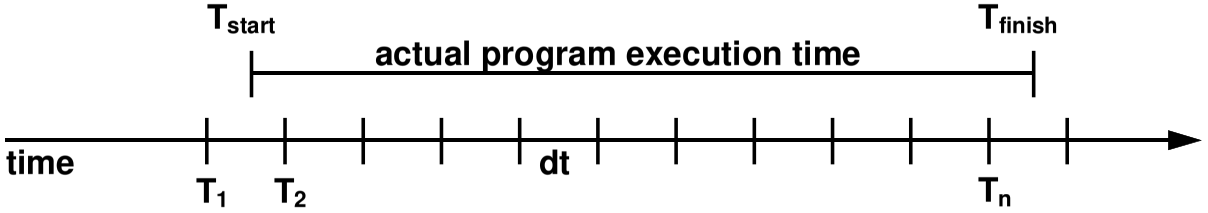
Anatomy of a timer



timer period: dt secs/tick

time resolution: 1/dt ticks/sec

Timer error



measured time: dtn

actual time: dt n + (T_{finish} - T_n) - (T_{start} - T_1)

f_{start} = (T_{start} - T_1)/dt (fraction overreported at start)

f_{finish} = (T_{finish} - T_n)/dt (fraction underreported at end)

error = dt f_{start} - dt f_{finish} = dt(f_{start} - f_{finish})

max error = plus/minus dt

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

Performance expressed as a time

Absolute time measures

- difference between start and finsh 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)

Performance expressed as a rate(cont)

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 matix-vector product vs nlogn 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:

Kernel	1d fft	sasum	saxpy	sdot	sgemm	sgemv	spvma
MFLOPS	8.5	3.2	6.1	10.3	6.2	15.0	8.1
%peak	11%	4%	7%	13%	8%	19%	10%

Relating time to system measures

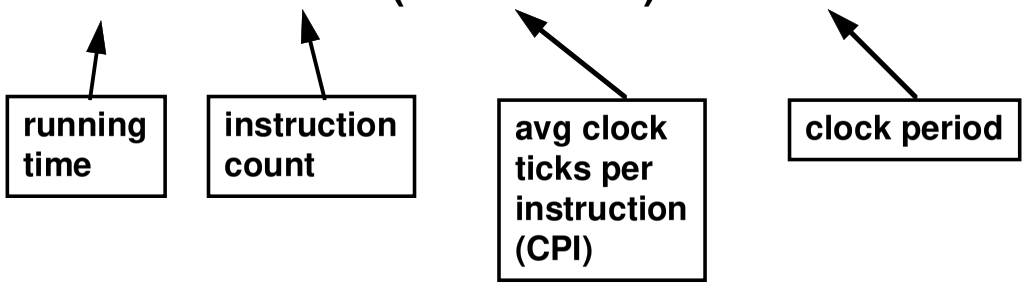
Suppose that for some program we have:

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

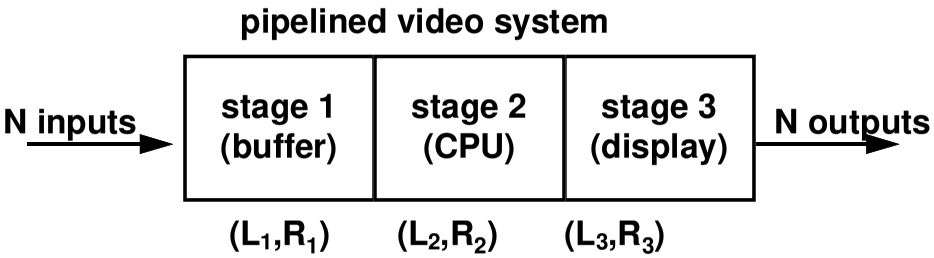
T secs = C ticks x P secs/tick

= (I inst/I inst) x C ticks x P secs/tick

T secs = I inst x (C ticks/inst) x P secs/tick



Relating time to latency and thruput



N images processed in T seconds

Latency: $L = L_1 + L_2 + L_3$ seconds to process one image
Thruput: $R = \min(R_1, R_2, R_3)$ images processed per second

T secs = L secs + (N-1 images x (secs/R images))
= L + (N-1)/R secs

Relating time to latency and thruput

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

- N = 1 => T = L
- N >> 1 => T = N-1/R

The stage with the smallest throughput is called the *bottleneck*.

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.

	time NY->Paris	total trip time	speedup over 747
747	8.5 hours	16.5 hours	1
SST	3.75 hours	11.75 hours	1.4

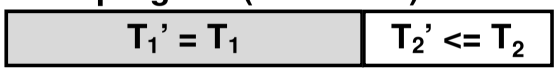
Taking the SST (which is 2.2 times faster) speeds up the overall trip by only a factor of 1.4!

Amdahl's law (cont)

Old program (unenhanced)

Old time: $T = T_1 + T_2$

New program (enhanced)



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

T_1 = time that can NOT be enhanced.

T_2 = time that can be enhanced.

T_2' = time after the enhancement.

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

Amdahl's law (cont)

Two key parameters:

$F_{enhanced} = T_2 / T$ (fraction of original time that can be improved)
$S_{enhanced} = T_2 / T_2'$ (speedup of enhanced part)

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

Amdahl's Law:

$$S_{overall} = T / T' = 1/((1-F_{enhanced}) + F_{enhanced}/S_{enhanced})$$

Key idea: Amdahl's law quantifies the general notion of diminishing returns. It applies to any activity, not just computer programs.

Amdahl's law (cont)

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):

	time NY->Paris	total trip time	speedup over 747
747	8.5 hours	16.5 hours	1
SST	3.75 hours	11.75 hours	1.4
rocket	0.25 hours	8.25 hours	2.0
rip	0.0 hours	8 hours	2.1

Amdahl's law (cont)

Useful corollary to Amdahl's law:

- $1 <= S_{overall} <= 1 / (1 - F_{enhanced})$

$F_{enhanced}$	Max $S_{overall}$	$F_{enhanced}$	Max $S_{overall}$
0.0	1	0.9375	16
0.5	2	0.96875	32
0.75	4	0.984375	64
0.875	8	0.9921875	128

Moral: It is hard to speed up a program.

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