Lecture #2metrics of performance, benchmarks

****
Review -- 1 min
****

Designing to last through trends

	CAPACITY	SPEED
Logic	2x in 3 years	2x in 3 years
DRAM	4x in 3 years	1.4x in 10 years
Disk	4x in 3 years	1.4x in 10 years

o changing fast
o at different rates
o --> different trade-offs

challenge -- don't bet on losing horse
opportunity -- new kinds of things we can do with computers

****
Outline - 1 min
****

Note: Review mode -- more topics than I normally want to cover in a day

Unifying idea: quantitative basis for architecture

Engineering Methodology
Bottom line: performance
benchmarking
summarizing peformance
Amdahl's law
cpu performance

****
Lecture - 20 min
****

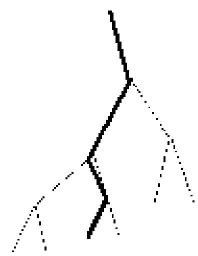
I. Engineering methodology
----

last time: architecture *quantitative*
claimed that made it strong field

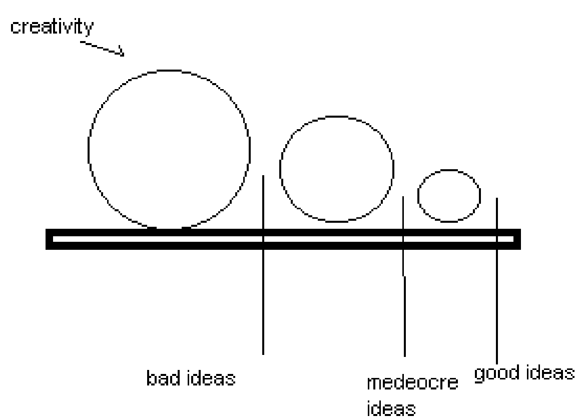
rigourous experimental approach that roots out bad ideas

would be nice to be able to write down a proof "here's god design"
not *proof*
    search space too large

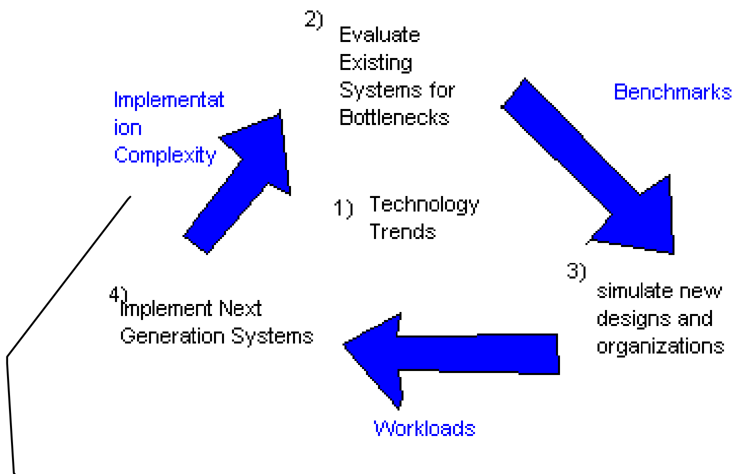
PICTURE:



another way of looking at it:



How does this process work?



One of hardest things to learn – importance of elegant, simple solutions
→ need to build

This class: Tools for doing this

- o benchmarks, traces, mixes
- o cost, delay, area, power
- o simulation
- o queuing theory
- o rules of thumb
- o fundamental laws

II. Bottom line: Performance (And cost)
----

today performance
cost next week

2 types of performance: Latency, Throughput

latency -- how long to do 1
throughput -- how many per unit of time

example: moving people from Austin to Dallas
compare Formula 1 race car with greyhound bus

latency: 1 hour v. 3 hours to get 1 person there
throughput: 1 person per hour v. 50 people per 3 hours

Which is faster?
Which is faster to get 50 people to Dallas?

Measure at different levels

Application	Answers per month operations per sec.
prgramming language	
compiler	
ISA	M's of insructions per second
Datapath	MB/s
Functional Units	
Transistors, wires, pins	cycles per second (MHz)

Level of measurement depends on what you're doing
- evaluating DRAM design -- MB/s?
- evaluating system -- answers per second?

****

Admin - 3 min

****

TA: Jing Zhao

No class monday -- holiday

HW 1 due: 1 week from today
- work in pairs (only)
- due 5pm friday (no late HW)
- available on line

Project topic selection: 2 weeks from today
- I'll discuss projects on wednesday

****

Lecture - 24 min
****

Comparing performance: Benchmarks

Rarely a dull event -- big $'s involved
--> charges and countercharges of "cheating"...

Patterson: "for better or worse, benchmarks shape a field"
--> if some number improves sales, focus engineering efforts on improving that number.
(whether or not it improves real-world performance)

    example -- compiler flags legal for only one program

    e.g. "don't worry about aliasing" --> makes it easier to allocate variables to registers

Types of benchmarks:

- o Marketing Metrics

  (simple: 1 number)

  MIPS - millions of instructions per second

QUESTION: What's wrong with mips?
A: ignores CPI, Instruction count
A: on what program?

  MFLOPS
  Same problems as MIPS
  + advertisers talk about "peak MFLOPS"

- o Toy bencmarks

  10-100 line program
  e.g. sieve, puzzle, quicksort, fibonacci

QUESTION: what's wrong with these?
A: no I/O, fits in cache, non-typical instruciton mixes/control patterns

o Synthetic benchmarks
attempt to match frequencies of real workloads
e.g. Dhrystone, whetstone

QUESTION: problems?
A: current processors depend on patern of instructions, not just individual instructions
A: defeated by/no credit to optimizing comilers
A: No I/O,

o kernels
key part of real program
Q: Problems
A: better, good for isolating performance features
A: still no I/O

o Real programs

best -- run your programs and see how they work
Problems?
Not good for marketing
Solution: suites

e.g. SPEC

story -- computer companies were having benchmark wars and accusing
one another of cheating
bad for whole industry

group anonymously got a set of real programs
every 3 years come out with new version
current version -- several floating point, several integer

(as people figure out how to cheat the benchmarks)

Benchmarking Games

- o different configurations to run same workload on 2 systems
- o compiler wired to optimize the workload
- o test spec biased towards one machine
- o arbitrary workload
- o small benchmark
- o benchmark manually translated to optimize performance

Common Benchmarking mistakes

- o only average behavior in test workload
  - average load on machine is about 0
  - you care about 98%-load
- o skewing of requests ignored
- o caching effects ignored
- o inaccurate sampling
  - e.g. when timer goes off -- take sample
  - timer interrupt lost when machine busy
- o ignoring monitoring overhead
- o not validating measurements
- o not ensuring same initial conditions
- o not measuring transient cold-start performance
- o using device utilizations for performance comparisons
  - machine 1 completes the benchmark with 25% cpu utilization
  - machine 2 completes the benchmark with 99% cpu utilization

    ? is it because machine 1 is 4 times faster?
    ? or because machine 1 is I/O limited and takes 4 times longer?

QUESTION: what is the right way to do this type of measurement?
A: increase the workload until both machines are saturated; report peak throughput?

o COLLECTING TO MUCH DATA BUT DOING TOO LITTLE ANALYSIS

How to summarize performance

"Faster Than"

X is n times faster than Y means

$$\frac{\text{Performance}(X)}{\text{Peformance}(Y)} = \frac{\text{Throughput}(X)}{\text{Throughput}(Y)} = \frac{\text{ExTime}(Y)}{\text{ExTime}(X)}$$

notice: peformance is inverse of ExTime

point is: this is a *convention* to save confusion

if A has 1 per second and B has 2 per second
could say "A is 50% of B" --> Speedup is 50%
"B is 100% faster than A" --> speedup is 100%

--> never say "slower than"

Mean -- how to summarize several numbers
Arithmetic/Harmonic -- track total execution time

arithmetic: 1/n * sum(time_1, time_2, ... time_n)

use harmonic for rates

harmonic: $$\frac{n}{\text{sum}(1/\text{rate}_1, 1/\text{rate}_2, \ldots 1/\text{rate}_n)}$$

example: suppose you send 10MB @ 1MB/s
then 10MB @ 5 MB/s
What is avg rate?

XXX: 1 + 5 / 2 = 3 MB/s WRONG

correct: first transfer took 10 seconds, second took 2 seconds --> total time 12 seconds for 20 MB --> avg rate 1.7 MB/s

(also weighted arithmetic, weighted harmonic)

Geometric

nth root of product of n samples

Arithmetic v. Geometric

Arithmetic: tracks time

Geometric: doesn't matter what machine you normalize to

Problem with geometric mean: encourages spending time to improve simplest programs v. improving programs where time is spent

e.g. 2 seconds --> 1 second gives same impact as 10000 seconds --> 5000 seconds
(and small programs easier to "crack")

Example: SPEC89 on IBM 550

	Ratio to VAX			Time		Weighted Time	
Program		Befo	After	Befr	After	Before	After
gcc		30	29	49	51	8.91	9.22
espreso		35	34	65	67	7.64	7.86
spice		47	47	510	510	5.69	5.69
doduc	46	49	41	38	5.81	5.45	
nasa7		78	144	258	140	3.43	1.86

li	34	34	183	183	7.86	7.86
eqntott	40	40	28	28	6.68	6.68
matrix300	78	730	58	6	3.43	0.37 <---!!!
fpppp	90	87	34	35	2.97	3.07
tomcatv	33	138	20	19	2.01	1.94
man	54	72	124	108	54.42	49.99
	Geometric		Aritmetic		Weighted Arithmetic	
	Ratio 1.33		1.16		1.09	

Story -- matrix300 spent >90% of its time on one line
cracked it -- got 10x improvement

moral -- geometric gives incentive to do unrealistic optimizations that crack benchmark

Amdahl's Law -- law of diminishing returns

Willie Sutton "Why do I rob banks? Cause that's where the money is."
You should do research same way!
(Depressing how many people don't!)

$$\text{speedup(enhancement)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{performance with}}{\text{performance without}}$$

suppose enhancement speeds up fraction F of program and leaves
rest unchanges?

ExTime new = ExTime old * (1-Fraction enhanced + FractionEnhaced/SpeedupEnhanced)

$$\text{Speedup overall} = \frac{1}{\text{1-fraction enhanced} + \dfrac{\text{fraction enhanced}}{\text{speedup enhanced)}}}$$

Question: suppose program spends 25% of its time doing floating point
What is MAX speedup I can get by improving floating point?
A: 1/0.75 = 1.33

CPU Performance

Remember TIME is a measure of performance
What you care about is how long does it take to run my program?

****
*** Wake Up! ***
*** Most useful equation in chapters 3 and 4 ***
****

CPU time = # instructions * cycles per instructin * clock cycle time

[time] = [instructions] * [cycles]/[instructions] * [time]/[cycle]

problem with "MIPS" and "MHz" as peformance metrics

Beware techniques that talk about improvments in one or 2 of the three only
e.g. optimizing compiler reduces number of instructions
increases cycles per instruction
Question: why?

****
Summay - 1 min
****

o Engineering Methodology
  technology trends
  measurements
o Bottom line: performance
  throughput or latency

- o benchmarking
  - "For better or worse, benchmarks shape a field"
  - --> want benchmarks s.t. improvements in benchmarks --> real-life improvements
    - (e.g. real programs)
- o summarizing peformance
  - "faster than"
  - means
- o Amdahl's law
  - law of dimishing returs
- o CPU performance "iron triangle"
  - CPU TIme = instr count * cycles per instruction * clock cycle time