Lecture #4

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

increase chip size --> much more expensive (^4)
low volums --> much more expensive
low gross margins --> where will research come from?

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

Basics of performance
CPU performance
intro to pipelining

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

Performance basics

1. comparing 2 alternatives
2. summarizing numbers
3. CPU performance

1. Comparing 2 alternatives
"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"

2. Summarizing numbers -- means

arithmetic v. harmonic

arithmetic: 1/n * sum(k_1, k_2, ... k_n)

use harmonic for rates

harmonic:
```
          n
   --------------
sum(1/k_1, 1/k_2, ... 1/k_n)
```

QUESTION: when do you use harmonic rather than arithmetic?
ANSWER: when comparing throughput rather than time

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

arithmetic v. geometric

geometric: nth root of product of n samples

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?
A: gets rid of "extra" instruction s--> remaining instructions more tightly packed --> harder to schedule dependencies

****
Admin - 3 min
****

HW1 due today at 5:00

Project poposals due next week
monday -- spend a few minutes matching people
that don't have partners

feel free to talk about project ideas with me before then
at ofic hours or other times

put name on back of pictures

****
Lecture - 24 min
****

Basic Pipelining

It's natural!

Laundry example

I have 4 loads
Washer: 30 minutes
Dryer: 40 minutes
Folding: 20 minutes

Sequential -- unpipelined

Load1: Wash Dry Fold
<---30--> <----- 40 -----> <---20--->
Load2: Wash Dry Fold
<--30--> <-----40-----> <--20-->

etc <draw rest on board>....

--> Total time for 4 loads -- 6 hours

Would never do it like that!!

Pipelined
Load1: Wash Dry Fold
<--30--><----40----> <--20->
Load2: Wash Dry Fold
<--30--><-10-> <----40----> <--20-->
Load3: Wash Dry Fold
<--30--><-10-> <----40----> <--20-->
Load4: Wash Dry Fold
<--30--><-10-> <----40----> <--
20-->

Total time: 30 + 40 + 40 + 40 +40 +20 = 210 = 3.5 hours

Laundry Lessons

1. Doesn't help latency of single task (helps throughput)
2. rate limited by slowest stage
3. multiple tasks operating in parallel
4. potential speedup = # pipe stages
5. unbalanced lenghts of stages reduce speedup
   (b/c rate limited by slowest)
6. time to "fill" and "drain" pipeline reduces speedup

<picture of DLX data path>

MIPS Instruction Set

32-bit fixed instruction format
32 registers (r0 contains 0)
3 address reg-reg ALU instructions
load/store
simple branch conditions

Register-Register
    Op Rs1 Rs2 Rd Opx--
Register-Immediate
    Op Rs1 Rd immediate------
Branch
    Op Rs1 Rs2/Opx immediate------
Jump/Call

Op   Target---------------------------------

<step through how each type of instruction works
using picture of data path and instruction types>

Unpipelined -- <CLOCK>
start an instruction
wait for logic to work itself out
<CLOCK> latch result in register
start next instruction

<picture of DLX datapath with pipeline registers>

<CLOCK> start I1
<CLOCK> start I2 ID I1
<CLOCK> ...

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

Basics of performance
CPU performance
intro to pipelining