Lab 1: Performance measurements and Amdahl's Law

Hung-Wei Tseng

Outline

- Performance equation
- Amdahl's Law
- Programming assignment — get familiar with C/C++

Q3-Q13: CPU Performance Equation

$$Performance = \frac{1}{Execution\ Time}$$

$$Execution\ Time = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

$$ET = IC \times CPI \times CT$$

- IC (Instruction Count) — Q3 — Q4
  - ISA, Compiler, algorithm, programming language, **programmer**
- CPI (Cycles Per Instruction)— Q7 — Q13
  - Machine Implementation, microarchitecture, compiler, application, algorithm, programming language, **programmer**
- Cycle Time (Seconds Per Cycle)— Q5 — Q6
  - Process Technology, microarchitecture, **programmer**

Q4: Performance equation (round 2)

- Consider the following c code snippet and x86 instructions implement the code snippet

C	x86

`for(i = 0; i < count; i++) {`
`    s += a[i];`
`}`
	`.L3:`
	`movslq  (%rdi), %rdx`
	`addq    $4, %rdi`
	`addq    %rdx, %tax`
	`cmpq    %rcx, %rdi`
	`jne     .L3`

Comparing the case where `count` equal to 1,000,000,000 and 2,000,000,000, what factor in performance equation would change?

A. IC
B. CPI
C. CT

Q4: Speedup per element?

- How much time do we have to spend on each s += a[i];

C	x86

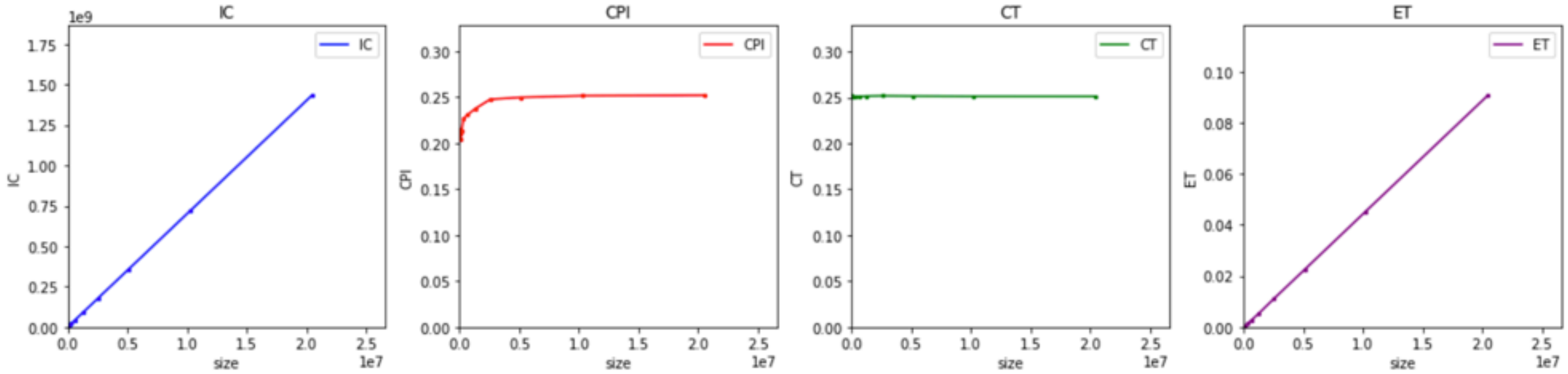
`for(i = 0; i < count; i++) {`
`    s += a[i];`
`}`
	`.L3:`
	`movslq  (%rdi), %rdx`
	`addq    $4, %rdi`
	`addq    %rdx, %tax`
	`cmpq    %rcx, %rdi`
	`jne     .L3`

Should be the same — theoretically

The reality is ...

size	ET	IC	CPI	MHz	CT
10000	0.000038	703477.840000	0.214835	3998.902357	0.250360
20000	0.000074	1403974.280000	0.209676	3966.218698	0.252225
40000	0.000144	2803346.000000	0.203906	3982.868429	0.251076
80000	0.000288	5604801.680000	0.204779	3989.110559	0.250683
160000	0.000598	11204254.480000	0.213047	3989.567204	0.250654
320000	0.001272	22408352.320000	0.226382	3988.729826	0.250707
640000	0.002592	44812633.200000	0.230514	3985.091346	0.250937
1280000	0.005343	89615660.920000	0.237185	3978.252816	0.251372
2560000	0.011165	179227083.720000	0.247559	3973.880677	0.251645
5120000	0.022505	358451482.800000	0.249731	3977.647497	0.251405
10240000	0.045290	716886493.640000	0.251519	3981.213906	0.251180
20480000	0.090740	1433751848.560000	0.252017	3982.006869	0.251130

Soon you will it's because memory!



Q14—Q16: Practicing Amdahl's Law

$$Speedup_{enhanced}(f, s) = \frac{1}{(1-f) + \frac{f}{s}}$$

f is the fraction of "execution time" — neither of the IC, CPI or CT

- Q14 — Q16: How to find the f in our program?

```
int main() {
    A();
    B();
    C();
    return 0;
}
```

function	time	fraction?
main	10	
A	5	50%
B	2	20%
C	3	30%

Q17—Q19: Throughput and latency

- Throughput is the amount of work/instructions/data that the system/machine/processor can deliver within a period of time
- It does not mean that the processor will deliver work at the "average" latency
- Throughputs reflect to performance relative well on large amount of work, but poorly on small amount of work

What's a "floating point operation"

- An arithmetic operation with at least one operand as "floating point" data types
- Memory access does not count as a FLOP
- How many floating point operations in the following statement?

```
uint32_t *A, size;
float *B;
for(int i = 0; i< size; i++) {
    B[i] = B[i] + A[i]*2;
}
```

floating point add

integer mul

Let's measure the FLOPS of matrix multiplications

```
double **a, **b, **c;
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

Floating point operations per second (FLOP"S"):

$$FLOPS = \frac{i \times j \times k \times 2}{ET_{seconds}}$$

Floating point operations (FLOP"s"):

$$i \times j \times k \times 2$$

Given $i = j = k = 2048$

$2^{3 \times 11} \times 2 = 2^{34}$ **FLOPs in total**

What if a and c are integers?

```
uint64_t **a, **b;
double **c;
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

Floating point operations (FLOP"s"):

$i \times j \times k \times 1$

Given $i = j = k = 2048$

$2^{3 \times 11} = 2^{33}$ **FLOPs in total**

Floating point operations per second (FLOP"S"):

$$FLOPS = \frac{i \times j \times k \times 1}{ET_{seconds}}$$

Demo: matmul on GPU

Size	Total FLOPs	Latency	Relative Latency	Throughput (Output Numbers Per	Relative Throughput
16x16x16	$(16)^3$x2				
32x32x32	$(32)^3$x2				
64x64x64	$(64)^3$x2				

Larger throughput doesn't mean shorter latency!

Demo: matmul on GPU

Size	Total FLOPs	Latency	Relative Latency	Throughput (Output Numbers Per Second)	Relative Throughput
16x16x16	$(16)^3$x2	~ 0.09ms	1	0.09ms/8192	1
32x32x32	$(32)^3$x2	~ 0.09ms	1	0.09ms/65536	8
64x64x64	$(64)^3$x2	~ 0.09ms	1	0.09ms/524288	64

Larger throughput doesn't mean shorter latency!

Programming assignment

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

Programming assignments

- Each lab will have a programming assignment in C/C++ for you to practice your skills of code optimizations
- escalab.org/datahub provides
  - VSCode server if you're more familiar with it
  - You may also use the editors in jupyterhub for code development
- The performance & grading are based on "gradescope's" server, not our servers.

Why C/C++ programming?

- The only pathway to performance programming

Use “gdb” as a debugging tool

- compile your program with “-g”
- gdb “binary_execution”
  gdb >> break hello2.cpp:”line_number”
  gdb >> run arguments

- gdb “binary_execution”
  gdb >> run arguments
  crashed!
  gdb >> bt

Lab 1: the main function & basic I/O

```
#include <fstream>
#include <iostream>

int main(int argc, char *argv[])
{
   std::ofstream ofs ("hello.txt", std::ofstream::out);
   ofs << "Hello CSE142L!\n";
   ofs.close();
   std::cout << "Execution Complete" << std::endl;
   return 0;
}
```

Hints to Lab 1 PA

- You need to manipulate the argv array
- You need to find out how to convert “ASCII” based characters into integers

Hints to Lab 1 PA

- You need to manipulate the argv array
- You need to find out how to convert “ASCII” based characters into integers

Questions for Lab 1 and PA 1?

Announcement

- Lab report 1 and PA 1 due 8/10 midnight through gradescope
- Lab 2 will be released on Sunday

Computer
Science &
Engineering

142L

つづく

How my “C code” becomes a “program”