CSE 142L: Practice what you learned from CSE142!

Hung-Wei Tseng

Goals

- Practice what you will learn in CSE142
- Extend what you will learn in CSE142
  - Understand deeply how a architecture affects performance
  - Understand deeply how to become a “performance programmer”

Course format

- 5 labs + 3 programming assignments
  - Due on every Sunday 11:59pm starting from 8/10
- In-Person Lectures —
  - Please check the schedule on https://calendar.google.com/calendar/u/0/r?cid=c_373ea7ba1adb25dcb44c3a3d1cb62af934f7601955381cdc89116d91596ba4af@group.calendar.google.com
  - Discussing current or upcoming labs
  - Like group office hours
- Youtube: https://www.youtube.com/profusagi
- No final exam

Grading & Schedule

Achievements	Final Grade	Due Date
Top 3 in PA #2 & #3	A+	
Programming Assignment #3	A	9/7/2025
Lab Report #5	A-	9/7/2025
Lab Report #4	B+	8/31/2025
Programming Assignment #2	B	8/24/2025
Lab Report #3	B-	8/24/2025
Lab Report #2	C+	8/17/2025
Programming Assignment #1	C	8/10/2025
Lab Report #1	C-	8/10/2025
	F	

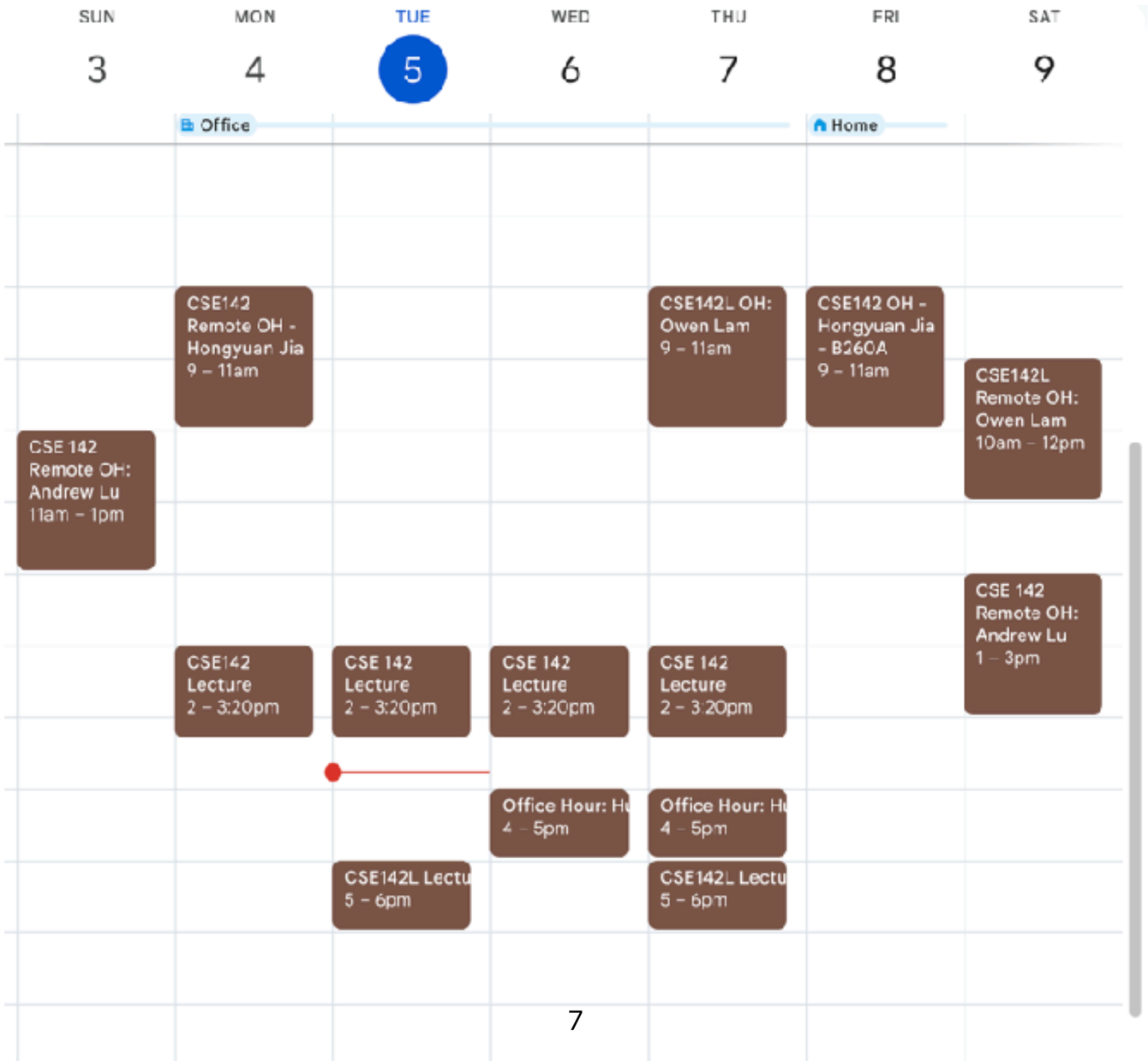
Instructor

- Instructor: Hung-Wei Tseng
  - Lectures
    TuTh 5:00p - 5:50p @ WCH 2205
    except for 8/15
  - Lab/Office hours:
    WTh 3:30-4:30p @ CSE 2210
  - Zoom: TH 8:30p-9:30p — check Google Calendar for the link

Tutor

- Andrew Lu
- Office hours:
  Sat 1p-3p @ Zoom
  Sun 11a-1p @ Zoom
- E-mail: cse142_su25 @ escalab.org
- Fun fact: I am a pokemon nerd
  (favorite pokemon is Greninja)

Office hour scheduling



Course resources

- Course webpage: https://www.escalab.org/classes/cse142l-2025su/
- Gradescope (for turning in lab reports and programming assignments) https://www.gradescope.com/courses/1068152
- Calendar for office/lab hours (you must login @ucsd.edu to view the schedule) https://calendar.google.com/calendar/u/0/embed?src=c_373ea7ba1adb25dcb44c3a3d1cb62af934f7601955381cdc89116d91596ba4af@group.calendar.google.com
- Discussion board (the same one as CSE142):
  - Search before ask
  - https://piazza.com/class/md99z1w7qks3cb

CSE142/CSE142L

- We have simplified the requirement
  - CSE142 and CSE142L's assignment and lab report is now the same document
  - Answer a few more questions and you can get the both done!
- You must be concurrently enrolled
  - You will not do well in 142L without being in 142 at the same time.
  - The content of CSE142L this summer has considered the 5-week nature in summer sessions
- Other common questions
  - Can I mix and match between 141/L and 142/L (e.g. I took 141 last quarter and I want to take 142L this quarter)
    - No! You cannot mix and match.
    - You must take either (141 and 141L) or (142 and 142L).
    - If you try, it will not count toward your degree.

Overview of a “Lab”

Not just for Lab 1, but almost the same for every lab

- Go to course home page: https://www.escalab.org/classes/cse142l-2025su/
- Click invitation link for the current lab
- Log into https://www.escalab.org/datahub
- Select the course you're taking — CSE142/L for our case
- Open a terminal
- Clone your starter repo.
- Open up assignment-lab.ipynb
- Follow the instructions in the notebook

GitHub

The Labs

- The Performance Equation
  - How can we measure performance?
  - What can we learn?
- Caches
  - Why is memory slow?
  - How can we write programs that access memory efficiently?
- Processors
  - How to exploit instruction-level parallelism?
  - How to optimize for branch and data dependencies?
- Parallelism
  - How threads and vectors improve performance?
  - Why aren’t they more helpful?

Course Infrastructure: Github and github classroom

- We will use github classroom to distribute starter code for the labs
- You'll use git/github to manage revisions etc.
- Git can be complex, but the basics are enough for this class.

Datahub @ escalab.org

Course Infrastructure: Docker

- All development and autograding takes places in a docker containers
- Containers provide
  - An isolated execution environment.
  - A reproduceable, consistent execution environment
  - A convenient way to bundle a set of tools together.
- The lab docker container provides everything you'll need for the labs.
- It also ensures that your code compiles/runs in the same way for you and for the autograder.

Course Infrastructure: Bare metal Servers in The Cloud

- We will do a lot of measurement in this class
  - Program performance
  - Program energy/power consumption
  - Detailed hardware behavior
- All of this requires "bare metal" servers
  - Bare metal – no virtualization
  - Full access to underlying processors (esp. performance counters)
- The Jupyter Notebook (and the autograder) run your code on some bare metal servers "in the cloud" (actually a cluster of computers hosted by my lab @ UCR)

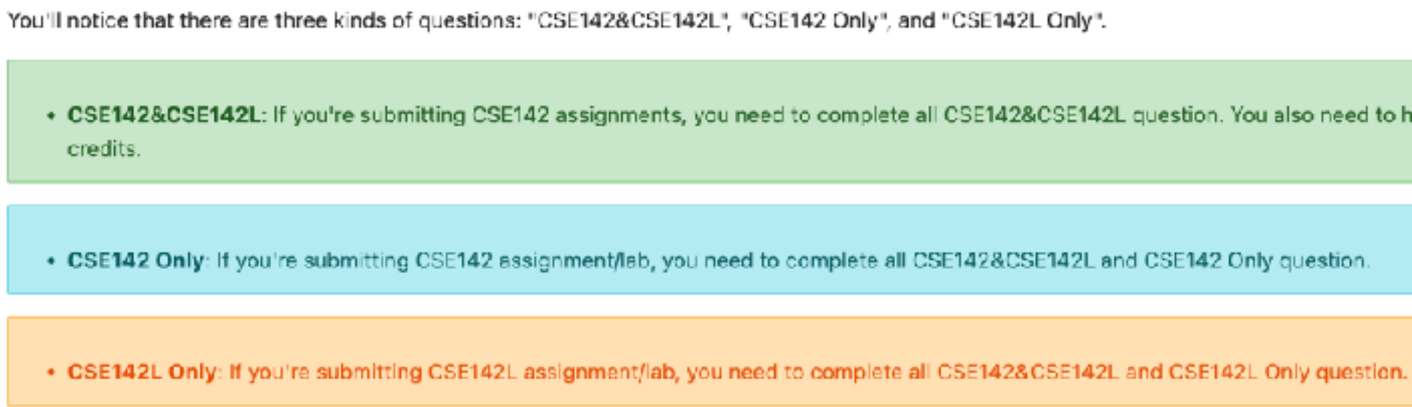
Jupyter Notebook

Course Infrastructure: Jupyter Notebook

- A large part of each lab is done in a Jupyter Notebook
  - Jupyter Notebook is a web-based, interactive computing environment
  - It's good for collecting and visualizing data
- If you haven't used Jupyter Notebook before...
  - That's fine. It's not that hard.
- We'll be accessing Jupyter Notebook via escalab.org's server.

Lab Report Questions

- There are about 20 questions per assignment/lab report
- You need to complete the following ones for CSE142L
- CSE142 & CSE142L (Green)
  - Demonstrate mastery
  - Give the right answer — earn points
- “CSE142L Only” (Orange)
  - ”forcing function” to get you to engage with the material
  - Give an answer — earn points



Let’s quickly walk through Lab 1

Programming assignment

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.

Lab 1: the profiling tool

- compile your program with “-g”
- perf record command > perf.data
  - Must run on the cloud as we want to profile the running instance on the cloud
- perf report to view the content

Why C/C++ programming?

- The only pathway to performance programming

The technical part behind assignment/lab 1

Classic CPU Performance Equation

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

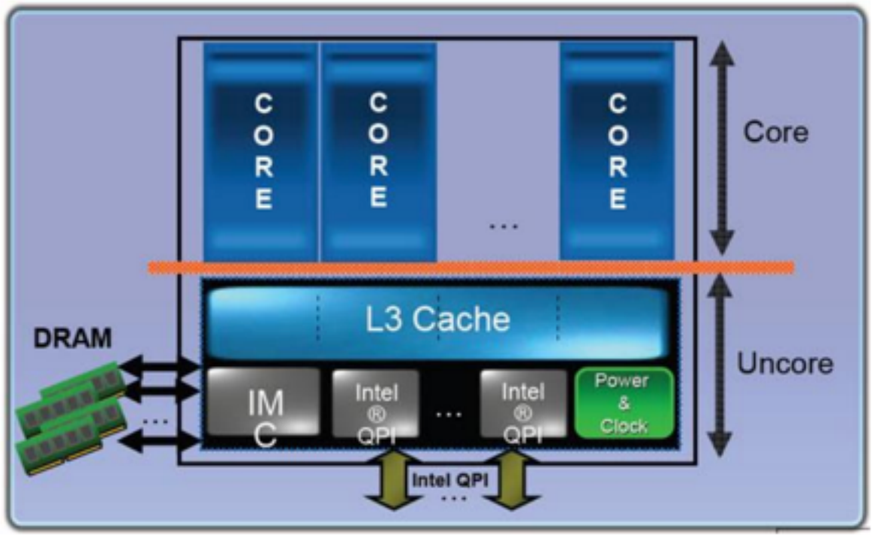
$$ET = IC \times CPI \times CT$$

$$= IC \times CPI \times \frac{1}{f}$$

How do I know which factor has room to improve?

Performance counters

- Intel®/AMD® processors mow have Performance Monitoring Units (PMUs) that can be programmed to count many performance-related events
  - One PMU per logical core (number of elapsed cycles, L1, L2 cache, TLB events, processed instructions, there are hundreds of events)
  - One in PMU uncore (L3 cache, memory controller, Intel® QPI events)



Programming PMUs

- Programming by reading/writing Model Specific Registers
- Much of hardware and events are platform specific
- Core PMU is enumerate in CPUID Leaf A:
  - Number of fully programmable counters (4 per logical core), a counter is assigned to count a certain event
  - Number of fixed function counters exist (3 per logical core): core clocks counter, reference clock counter, instruction counter
- Some uncore and core programmable counters can be only programmed with certain types of events
- Other tricky restrictions apply, restrictions are documented in the event list

How CSE142L uses performance counters

```
enum {
    COUNTER_CPU_CYCLES,          // NUMBER OF CYCLES
    STALLED_CYCLES_FRONTEND,
    STALLED_CYCLES_BACKEND,
    INSTRUCTIONS,                // NUMBER OF INSTRUCTIONS (IC)
    DTLB_ACCESSES,
    DTLB_MISSES,
    DL1_LOAD_ACCESSES,
    DL1_LOAD_MISSES,
    DL1_STORE_ACCESSES,
    DL1_STORE_MISSES,
    DL1_PREFETCH_ACCESSES,
    DL1_PREFETCH_MISSES,
};


static struct perf_event_attr attrs[] = {
    { .type = PERF_TYPE_HARDWARE, .config = PERF_COUNT_HW_CPU_CYCLES},
    { .type = PERF_TYPE_HARDWARE, .config = PERF_COUNT_HW_STALLED_CYCLES_FRONTEND},
    { .type = PERF_TYPE_HARDWARE, .config = PERF_COUNT_HW_STALLED_CYCLES_BACKEND},
    { .type = PERF_TYPE_HARDWARE, .config = PERF_COUNT_HW_INSTRUCTIONS},
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_DTLB | (PERF_COUNT_HW_CACHE_OP_READ << 8) | (PERF_COUNT_HW_CACHE_RESULT_ACCESS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_DTLB | (PERF_COUNT_HW_CACHE_OP_READ << 8) | (PERF_COUNT_HW_CACHE_RESULT_MISS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_READ << 8) | (PERF_COUNT_HW_CACHE_RESULT_ACCESS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_READ << 8) | (PERF_COUNT_HW_CACHE_RESULT_MISS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_WRITE << 8) | (PERF_COUNT_HW_CACHE_RESULT_ACCESS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_WRITE << 8) | (PERF_COUNT_HW_CACHE_RESULT_MISS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_PREFETCH << 8) | (PERF_COUNT_HW_CACHE_RESULT_ACCESS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_PREFETCH << 8) | (PERF_COUNT_HW_CACHE_RESULT_MISS << 16) },
    };
```

How CSE142L uses performance counters (cont.)

```
static inline int
sys_perf_event_open(struct perf_event_attr *attr,
                    pid_t pid, int cpu, int group_fd,
                    unsigned long flags)
{
    attr->size = sizeof(*attr);
    return syscall(__NR_perf_event_open, attr, pid, cpu,
                   group_fd, flags);
}


void perfstats_init(void)
{
    int pid = getpid();
    int i;

    for (i = 0; i < STAT_COUNT; i++) {
        attrs[i].inherit = 1;
        attrs[i].disabled = 1;
        attrs[i].exclude_kernel = 0;
        attrs[i].enable_on_exec = 0;
        fds[i] = sys_perf_event_open(&attrs[i], pid, -1, -1, 0);
//        fprintf(stderr,"PC: %d %d %X\n",i, fds[i], attrs[i].config);
    }
}


void perfstats_enable(void)
{
    int i;
    for (i = 0; i < STAT_COUNT; i++) {
        if (fds[i] <= 0)
            continue;

        ioctl(fds[i], PERF_EVENT_IOC_ENABLE);
    }
    for (i = 0; i < STAT_COUNT; i++) {
        if (fds[i] > 0)
            read(fds[i], &performance_counters[i], sizeof(performance_counters[i]));
    }
    gettimeofday(&time_start, NULL);
}
```

How CSE142L uses performance counters

```
enum {
    COUNTER_CPU_CYCLES,          // NUMBER OF CYCLES
    STALLED_CYCLES_FRONTEND,
    STALLED_CYCLES_BACKEND,
    INSTRUCTIONS,                // NUMBER OF INSTRUCTIONS (IC)
    DTLB_ACCESSES,
    DTLB_MISSES,
    DL1_LOAD_ACCESSES,
    DL1_LOAD_MISSES,
    DL1_STORE_ACCESSES,
    DL1_STORE_MISSES,
    DL1_PREFETCH_ACCESSES,
    DL1_PREFETCH_MISSES,
};
```

$$CPI = \frac{\#\ of\ cycles}{IC}$$

$$CT = \frac{ET}{\#\ of\ cycles}$$

```
static struct perf_event_attr attrs[] = {
    { .type = PERF_TYPE_HARDWARE, .config = PERF_COUNT_HW_CPU_CYCLES},
    { .type = PERF_TYPE_HARDWARE, .config = PERF_COUNT_HW_STALLED_CYCLES_FRONTEND},
    { .type = PERF_TYPE_HARDWARE, .config = PERF_COUNT_HW_STALLED_CYCLES_BACKEND},
    { .type = PERF_TYPE_HARDWARE, .config = PERF_COUNT_HW_INSTRUCTIONS},
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_DTLB | (PERF_COUNT_HW_CACHE_OP_READ << 8) | (PERF_COUNT_HW_CACHE_RESULT_ACCESS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_DTLB | (PERF_COUNT_HW_CACHE_OP_READ << 8) | (PERF_COUNT_HW_CACHE_RESULT_MISS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_READ << 8) | (PERF_COUNT_HW_CACHE_RESULT_ACCESS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_READ << 8) | (PERF_COUNT_HW_CACHE_RESULT_MISS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_WRITE << 8) | (PERF_COUNT_HW_CACHE_RESULT_ACCESS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_WRITE << 8) | (PERF_COUNT_HW_CACHE_RESULT_MISS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_PREFETCH << 8) | (PERF_COUNT_HW_CACHE_RESULT_ACCESS << 16) },
    { .type = PERF_TYPE_HW_CACHE, .config = PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_PREFETCH << 8) | (PERF_COUNT_HW_CACHE_RESULT_MISS << 16) },
    };
```

How do I know the capability of my processors? (In Linux)

- lscpu
- cat /proc/cpuinfo
- cpupower frequency-info -n

And … Google!

https://www.intel.com/content/www/us/en/products/sku/120485/intel-xeon-gold-6140-processor-24-75m-cache-2-30-ghz/specifications.html

Essentials

Downlc

Product Collection	Intel® Xeon® Scalable Processors
Code Name	Products formerly Skylake
Vertical Segment	Server
Processor Number	6140
Lithography	14 nm
Recommended Customer Price	$2445.00-$2451.00

CPU Specifications

Total Cores	18
Total Threads	36
Max Turbo Frequency	3.70 GHz
Processor Base Frequency	2.30 GHz
Cache	24.75 MB L3 Cache
Max # of UPI Links	3
TDP	140 W

intel®

Intel® 64 and IA-32 Architectures Optimization Reference Manual

The processor can run at 3.7 GHz

What can we improve?

[11]:
```
display_mono(render_csv("first.csv"))
```

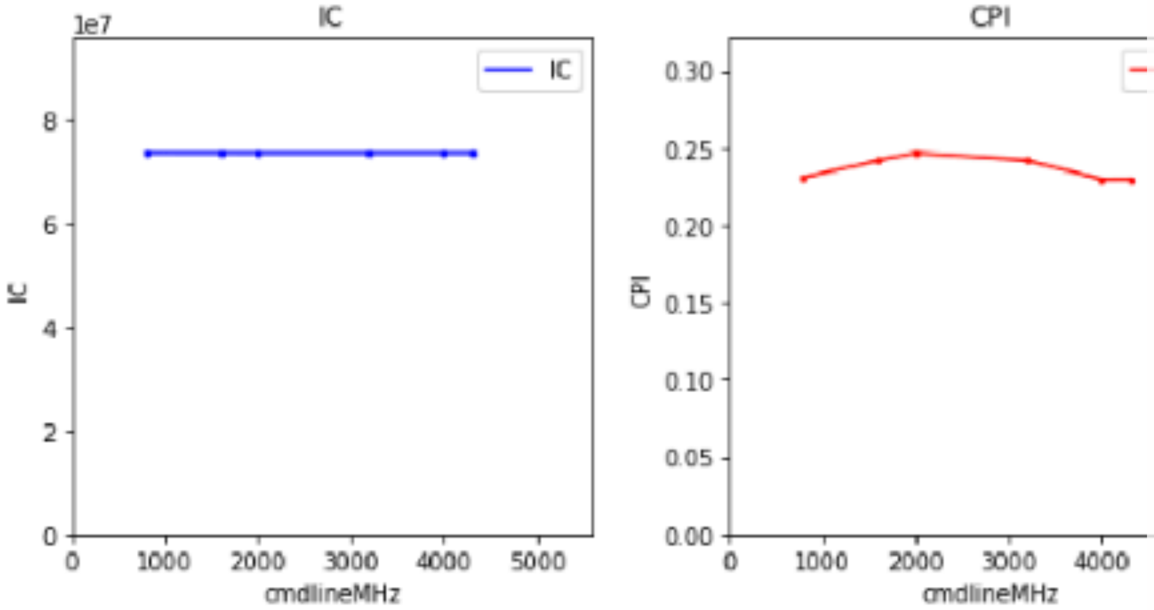
	size	rep	function	IC	Cycles	CPI	MHz	CT	ET	cmdlineMHz
0	1024	0	baseline_int	78322	39990	0.510585	5712.857143	0.175044	0.000007	3300
1	1024	1	baseline_int	75026	19271	0.256858	3211.833333	0.311349	0.000006	3300
2	2048	0	baseline_int	146706	32974	0.224762	3297.400000	0.303269	0.000010	3300
3	2048	1	baseline_int	146706	32579	0.222070	3257.900000	0.306946	0.000010	3300

CPI can be better

MHz can be higher!

Q5: play with the cycle time

cmdlineMHz	ET	IC	CPI	MHz	CT
800	0.021255	73444420.980000	0.230694	797.135752	1.254492
1600	0.011182	73427339.660000	0.241837	1588.053811	0.629707
2000	0.009124	73420051.300000	0.246631	1984.660133	0.503870
3200	0.005575	73415018.480000	0.241858	3185.149227	0.313960
4000	0.004232	73414207.100000	0.229542	3981.747969	0.251149
4300	0.004118	73413700.400000	0.229615	4093.551989	[illegible]



What's the speedup of 4.3 GHz v.s. 3.2 GHz?

$$Speedup = \frac{0.005575}{0.004118} = 1.35381253$$

This is the type of answer that the autograder expected to give credits

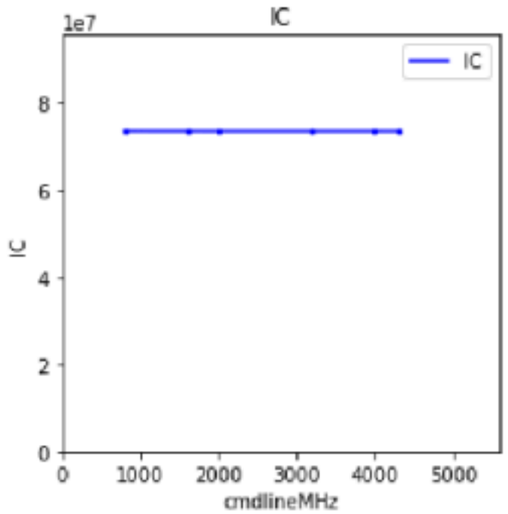
- **Don't round, just copy-and-paste**
- **Must match the numbers you got — don't copy-paste from others**

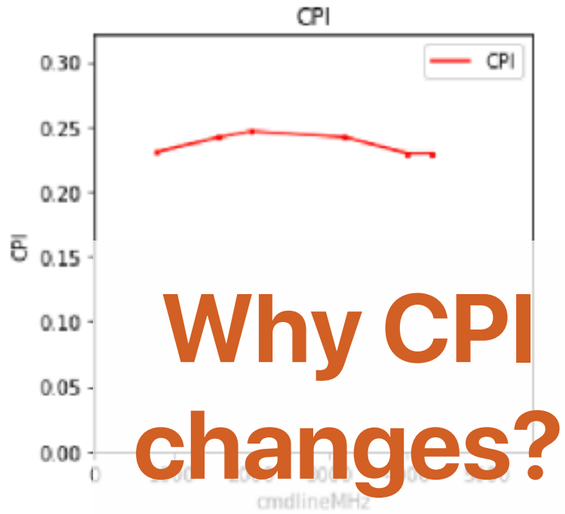
Q5: play with the cycle time

cmdlineMHz	ET	IC	CPI	MHz	CT
800	0.021255	73444420.980000	0.230694	797.135752	1.254492
1600	0.011182	73427339.660000	0.241837	1588.053811	0.629707
2000	0.009124	73420051.300000	0.246631	1984.660133	0.503870
3200	0.005575	73415018.480000	0.241858	3185.149227	0.313960
4000	0.004232	73414207.100000	0.229542	3981.747969	0.251149
4300	0.004118	73413700.400000	0.229615	4093.551989	0.244314

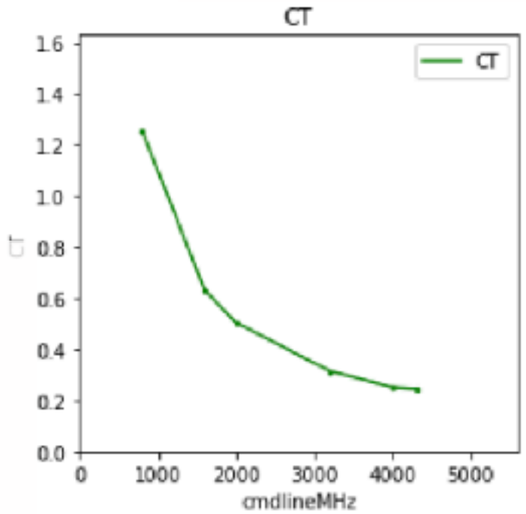
What changed?

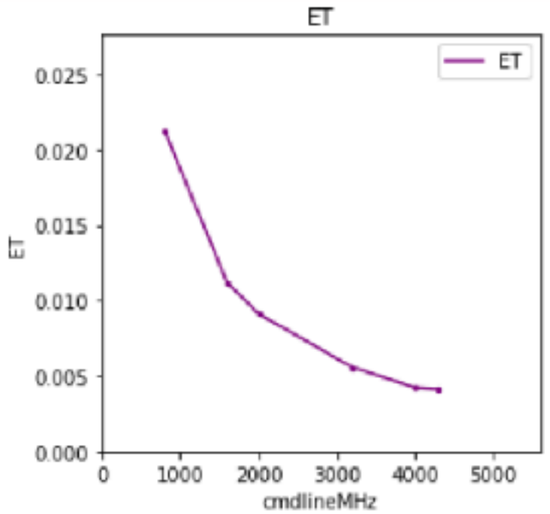
What did not change?





Why CPI changes?





What can we improve?

cmdlineMHz	ET	IC	CPI	MHz	CT
800	0.021255	73444420.980000	0.230694	797.135752	1.254492
1600	0.011182	73427339.660000	0.241837	1588.053811	0.629707
2000	0.009124	73420051.300000	0.246631	1984.660133	0.503870
3200	0.005575	73415018.480000	0.241858	3185.149227	0.313960
4000	0.004232	73414207.100000	0.229542	3981.747969	0.251149
4300	0.004118	73413700.400000	0.229615	4093.551989	0.244314

CPI can be better

What can change the CPI?

- We need to change the "percentages" of "instruction types"

$$ET = (5 \times 10^9) \times (20\% \times 4 + 20\% \times 3 + 60\% \times 1) \times \frac{1}{4 \times 10^9}\ sec = 2.5\ sec$$

total # of dynamic instructions

average CPI

- How can we change that?
  - Programmers: Q7 — Q9, Q12—Q13
  - Compilers: Q10 — Q11

Q13: A is better than B!

- gcc has different optimization levels.
  - -O0 — no optimizations
  - -O3 — typically the best-performing optimization

A

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

B

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

Data types in “x86 instructions” vs “C/C++”

C declaration	x86	x86 instruction suffix	x86-64 Size (Bytes)	functional unit
char	Byte	b	1	Integer
short	Word	w	2	
int	Double word	l	4	
unsigned	Double word	l	4	
long int	Quad word	q	8	
unsigned long	Quad word	q	8	
char *	Quad word	q	8	
float	Single precision	s	4	floating point units
double	Double precision	d	8	
long double	Extended precision	t	16	

MOV and addressing modes

- MOV instruction moves data between registers/memory
- MOV instruction has many address modes

instruction	meaning	arithmetic op	memory op
`movl $6, %eax`	`R[eax] = 0x6`	1	0
`movl .L0, %eax`	`R[eax] = .L0`	1	0
`movl %ebx, %eax`	`R[ebx] = R[eax]`	1	0
`movl -4(%ebp), %ebx`	`R[ebx] = mem[R[ebp]-4]`	2	1
`movl (%ecx,%eax,4), %eax`	`R[eax] = mem[R[ebx]+R[edx]*4]`	3	1
`movl -4(%ecx,%eax,4), %eax`	`R[eax] = mem[R[ebx]+R[edx]*4-4]`	4	1
`movl %ebx, -4(%ebp)`	`mem[R[ebp]-4] = R[ebx]`	2	1
`movl $6, -4(%ebp)`	`mem[R[ebp]-4] = 0x6`	2	1

Arithmetic Instructions

- Operands can come from either registers or a memory location

instruction	meaning	arithmetic op	memory op
`subl $16, %esp`	`R[%esp] = R[%esp] – 16`	1	0
`subl %eax, %esp`	`R[%esp] = R[%esp] – R[%eax]`	1	0
`subl –4(%ebx), %eax`	`R[eax] = R[eax] – mem[R[ebx]–4]`	2	1
`subl (%ebx, %edx, 4), %eax`	`R[eax] = R[eax] – mem[R[ebx]+R[edx]*4]`	3	1
`subl –4(%ebx, %edx, 4), %eax`	`R[eax] = R[eax] – mem[R[ebx]+R[edx]*4–4]`	3	1
`subl %eax, –4(%ebx)`	`mem[R[ebx]–4] = mem[R[ebx]–4]–R[eax]`	3	2

The CPI of each instruction is different!

IDIV (M16)	[12;20]	
IDIV (M32)	[11;19]	
IDIV (M64)	[14;23]	
IDIV (M8)	[17;22]	
IDIV (R16)	[11;16]	
IDIV (R32)	[10;15]	
IDIV (R64)	[14;18]	
IDIV (R8h)	[17;39]	
IDIV (R8l)	17	
IMUL (M16)	[0;10]	
IMUL (M32)	[3;9]	
IMUL (M64)	[3;9]	
IMUL (M8)	[3;8]	
IMUL (R16)	[0;5]	
IMUL (R16, M16)	[3;8]	
IMUL (R16, M16, 0)	[0;9]	
IMUL (R16, M16, I16)	[0;9]	
IMUL (R16, M16, I8)	[0;9]	
IMUL (R16, R16)	3	

https://uops.info/table.html

References

- David Levinthal "Performance Analysis Guide for Intel® Core™ i7 Processor and Intel® Xeon™ 5500 processors" http://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf
- Intel® 64 and IA-32 Architectures Software Developer‟s Manual, Volume 3B: System Programming Guide, Part 2 http://www.intel.com/products/processor/manuals/
- Intel® Xeon® Processor 7500 Series Uncore Programming Guide http://www.intel.com/Assets/en_US/PDF/designguide/323535.pdf
- Peggy Irelan and Shihjong Kuo "Performance Monitoring Unit Sharing Guide " http://software.intel.com/file/20476

Submission!

Submission: Gradescope

- All assignments will mostly be submitted via gradescope
- Lab report: Jupyter Notebook
  - Autograded
  - Submitted via github
- Programming assignments
  - Autograded
  - Submitted via github
- Post-lab survey
  - Embedded in the lab as a google form.

Announcement

- Lab report 1 and programming assignment 1 due this Sunday
- Please make sure that you can login escalab.org/datahub as early as possible

Computer
Science &
Engineering

142L

つづく