Why Computer Systems Programming?

2024 Fall ECE454: Computer Systems Programming — Lecture 1

Jon Eyolfson

1.0.0

I’m Jon, Your Instructor

Eyolfson

I’m Jon, Your Instructor

Eyolfson

I’m Jon, Your Instructor

Elf

I’m Jon, Your Instructor

Elf   son

I'm Jon, Your Instructor

E lfson

I'm Jon, Your Instructor

Eyolfson

Why Computer Systems Programming?

You can write functional software, but can you write performant software?

Most software you interact with runs on cloud servers that:

- Are very expensive, saving 2% of the costs is significant
- Use a lot of power
- Are idle a significant amount of time

Important URLs for Course Resources

Very Important: Sign into `https://compeng.gg`

- Lectures: `https://eyolfson.com/courses/ece454/`
- Labs: `https://compeng-gg.github.io/2024-fall-ece454-docs/`
- Materials: `https://github.com/compeng-gg/2024-fall-ece454-materials`

These links and others are on: `https://q.utoronto.ca/` (Quercus)

Labs on GitHub, Discussion on Discord, Streams on YouTube

GitHub

Discord

YouTube

Connect your Discord and GitHub on `https://compeng.gg`

Anonymous Discord Messages

Some students don’t want to ask questions on Discord because it’s not anonymous, **we fixed that**

Use the command:

```
/anon <message>
```

The command sends your message anonymously in the current channel

Lecture Attendance is Still Important

It's much faster to get feedback from you and clarify if anything is unclear

We'll have live coding, I'll be able to explain any happy accidents

Let me know anything else that might make the course better!

This is a new course for me, so be gentle

Evaluation for this Course

Assessment	Duration	Weight	Due Date
Lab 1	1 week	5%	September 16
Lab 2	3 weeks	9%	October 7
Midterm Exam	1.25 hours	20%	October 17 (tentative)
Lab 3	3 weeks	9%	November 5
Lab 4	2 weeks	7%	November 19
Lab 5	2 weeks	10%	December 3
Final Exam	2.5 hours	40%	TBD

Academic Honesty Policy

You can study together, discuss concepts on Discord

Don’t post lab code on Discord, any other code is okay

Any cheating is not tolerated, and will only hurt you

Please Provide Feedback!

This course is challenging, please let me know if anything is unclear

You can ask any questions, there’s lots of open source software to look at too

By the end of the course you’ll be a better programmer

We Care About Performance in This Course

Specifically, we'll focus on:

- Scalability
- Efficiency

In that Past, You Could Just Wait

Moore’s Law was in full swing doubling transistors every ~18 months, and all the extra transistors went to a single core speed

That meant if you wanted your program to go twice as fast, just wait 18 months and buy new hardware

Current Trends

Moore’s law hasn’t hit a wall just yet (although physical limits are coming)

Single core performnce has leveled off, due to physics a CPU
- running at 10 GHz would be hotter than the surface of the sun
  - (don’t quote me on that)

Modern systems use many cores, and you need to use them
- You’ll need to program with parallelism in mind

For large scale software, you’ll also have to use multiple machines

There's Also a Trend to Specialized Hardware

Most CPUs now have dedicated hardware certain functions
- e.g. video encoding/decoding

ASIC (application-specific integrated circuit) for less common functions
- e.g. bitcoin mining

You can also program GPUs that are massively parallel

Latency Numbers Every Programmer Should Know

L1 cache reference (~ 80 KB)	1	ns
Branch mispredict	3	ns
L2 cache reference (~ 2 MB)	4	ns
Mutex lock/unlock	17	ns
Send 1K bytes over 1 Gbps network	44	ns
Main memory reference	100	ns
Compress 1K bytes with Zippy	2	µs
Read 1 MB sequentially from memory	3	µs
Read 4K randomly from SSD	16	µs
Read 1 MB sequentially from SSD	49	µs
Round trip within same datacenter	500	µs
Read 1 MB sequentially from disk	825	µs
Disk seek	2	ms
Send packet Cali → Netherlands → Cali	150	ms

Reducing Latency Requires You to Find the Bottleneck

If you can avoid disk I/O, you could improve performance by 100 000 times!
- Memcached for example, caches requests to a database or API

You could also allocate your memory in a better way
- If you can fit the data entirely in cache, it’ll be much faster

This makes single-threaded applications faster, but we’ll also parallelize

We’ll start with single-threaded applications and than multi-threaded

Module 1: Code Measurement

Topics

Finding the bottleneck

Principles of code optimization, and using an optimizing compiler

Profiling (measuring time)

Labs

Lab 1: investigating Lab 2

Module 2: Memory Management

Topics

- Memory hierarchy
- Caches and locality
- Virtual memory

Labs

- Lab 2: Optimizing memory performance
- Lab 3: Writing your own memory allocator

Module 3A: Parallelization on a Single Machine

Topics

Threads and threaded programming

Synchronization and performance

Labs:

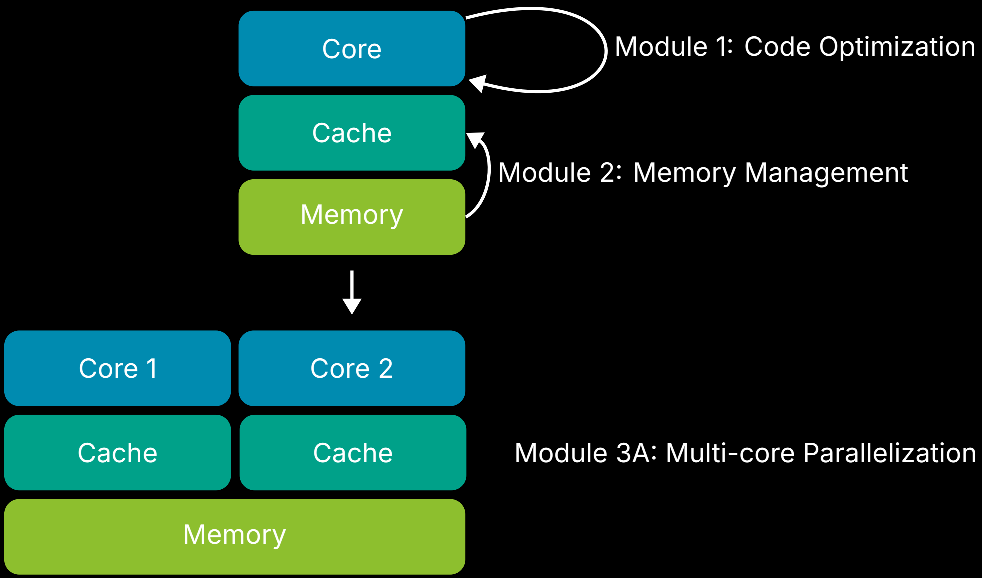
Lab 4: Threads and synchronization methods
Lab 5: Parallelizing a game simulation program

Module 3B: Parallelization on Mutliple Machines

Topics

Frameworks for big data analytics
Cloud computing and storage systems

How the Modules Fit Together



Code Examples are Available to You

You should have access to the GitHub repository:
`https://github.com/compeng-gg/2024-fall-ece454-materials`

The setup is the same as the labs:
`https://compeng-gg.github.io/2024-fall-ece454-docs/setup/`

You should just have to open the folder in VSCode

Unlike the labs, these are your steps to compile (in the `examples` directory):

```
meson setup build
meson compile -C build
```

(the executables will be in the `build` directory)

"Premature optimization is the root of all evil."

- Sir Tony Hoare

Let's Try to Write a `cp` Clone

The source file is located in the `materials` repository at:
  `lectures/01-why-computer-systems-programming/examples/cp-clone.c`

At the comment, we might add the following optimization:

```c
if (stat.st_size == 0) {
  return 0;
}
```

so we can avoid any unnecessary `read` calls

Let’s Try to Write a `cp` Clone

The source file is located in the `materials` repository at:

`lectures/01-why-computer-systems-programming/examples/cp-clone.c`

At the comment, we might add the following optimization:

```
if (stat.st_size == 0) {
  return 0;
}
```

so we can avoid any unnecessary `read` calls

However, this breaks copying “files” like `/proc/cpuinfo`

How Do You Know an Optimization is Premature?

This is a bit trickier to answer...

What's the purpose of the program? Are you only going to use it once?
- It likely doesn't matter

Are you optimizing the bottleneck?
- If not, you don't need to waste time

Are you optimizing the common case or special case?
- If it's the special case, you don't need to worry

What price am I paying?
- Develop productivity? Readability? Program size?

What You Want to Avoid

```
// When I wrote this, only God and I understood what I was doing
// Now, God only knows
//
// Therefore if you are trying to optimize this routine and it fails
// (most surley), please increase this counter as a warning for the
// next person:
//
// total_hours_wasted_here = 254
```

The Easiest Way to Measure Performance—Speedup

We're used to latency, how long does it take to do a task

$S_{\text{latency}} = \frac{L_{\text{old}}}{L_{\text{new}}}$

where,

S_{latency} is the speedup (in terms of latency)
L_{old} is the latency of the old task
L_{new} is the latency of the new task

The Easiest Way to Measure Performance—Speedup

We're used to latency, how long does it take to do a task

$S_{\text{latency}} = \frac{L_{\text{old}}}{L_{\text{new}}}$

where,
S_{latency} is the speedup (in terms of latency)
L_{old} is the latency of the old task
L_{new} is the latency of the new task

For example: our original task takes 1 second to complete, then we optimize it to 0.5 seconds $L_{\text{old}} = 1s$

$L_{\text{new}} = 0.5s$

then, $S_{\text{latency}} = \frac{L_{\text{old}}}{L_{\text{new}}} = \frac{1s}{0.5s} = 2$

You Can Measure the Latency of an Exectable with `time`

You can use `/usr/bin/time -p <executable>` (not `time` the shell built-in), an example of the output is:

```
real 159.98
user 0.21
sys 3.26
```

`real` is the number of seconds that actually passed (wall time)

`user` is the number of seconds all cores execute for in user mode (for multicore this can be greater than `real`)

`sys` is the number of seconds all cores execute for in kernel mode

We Also Want to Consider Throughput

Throughput is the number of tasks you can do per unit time

$$Q = \frac{P}{L}$$

where,
Q is the throughput
P is the number of tasks you can do simultaneously
L is the latency of a single task

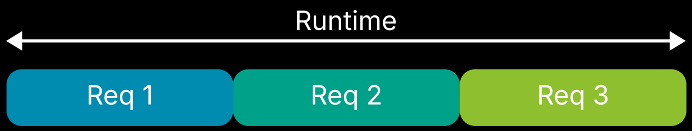
Airplanes as a Real Life Example

Plane	YYZ → FRA	Passengers
Airbus	8 hours	470
Concorde	4 hours	132

Which plane has better performance?

Any other real life examples you can think of?

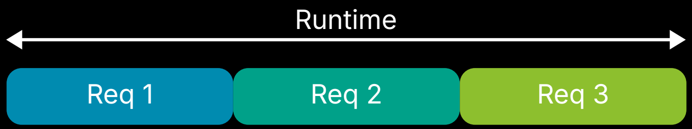
Let’s Consider a Webserver Handling Requests Serially



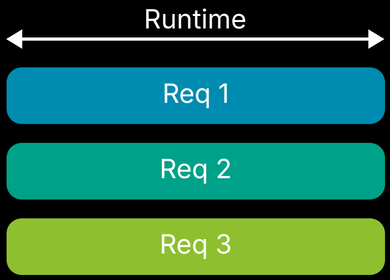
Assume we buy a faster CPU, we’ll have higher throughput and lower latency

This is a positive correlation

Now the Webserver Handles Requests Parallelly



turns into



Latency is worse (larger), but throughput is higher

This is a negative correlation

There are Limits—Amdahl’s Law

The law states that overall speedup you achieve depends on the fraction you can speedup

Let p be the proportion of the task you can speedup
Let s be the speedup of that proportion

$S_{\text{latency}}(s) = \frac{1}{(1-p)+\frac{p}{s}}$

There are Limits—Amdahl's Law

The law states that overall speedup you achieve depends on the fraction you can speedup

Let p be the proportion of the task you can speedup
Let s be the speedup of that proportion

$S_{\text{latency}}(s) = \frac{1}{(1-p)+\frac{p}{s}}$

which means:

$$\begin{cases} S_{\text{latency}}(s) \leq \dfrac{1}{1-p} \\ \lim\limits_{s\to\infty} S_{\text{latency}}(s) = \dfrac{1}{1-p}. \end{cases}$$

In other words, if you can speed up 90% of your code ($p = 0.9$), the best overall speed you can achieve is 10

An Example Using Amdhal's Law

If an optimization makes loops go 3 times faster, and my program spends 70% of its time in loops, how much faster will my program go?

There’s Other PerformanceConsiderations

Utilization, Goodput (only measure useful data), Jitter (consistency)

Do we care about the best case, worse case, or average case?

Performance evlauation is an active field of research

What Being Responsible for Performance May Do to You

Experience

Goose farmer
Self-employed
Jul 2023 - Present · 1 yr 2 mos
On-site

Microsoft
22 yrs 4 mos

- **Principal Performance Architect**
  Full-time
  Jul 2022 - Jul 2023 · 1 yr 1 mo
  Chehalis, Washington, United States · Remote

- **Principal Software Development Engineer**
  Apr 2001 - Jul 2022 · 21 yrs 4 mos
  Redmond,WA

We’ll Use C++ in the Course

Why would you want to use C++? The most illustrative example is sorting

What’s faster, and why? What is easier to read?

1. Using `qsort` in C
2. Using a hand written sort in C
3. Using `sort` in C++ with an array
4. Using `sort` in C++ with a `std::vector`