ECE 459

Programming for Performance

Winter 2023

Jeff Zarnett, Patrick Lam & **Huanyi Chen**

jzarnett@uwaterloo.ca, patrick.lam@uwaterloo.ca,

huanyi.chen@uwaterloo.ca

UNIVERSITY OF WATERLOO

About the Course

- Profiling computer systems; bottlenecks, Amdahl's law.
- Concurrency: threads and locks.
- Techniques for programming multicore processors; cache consistency.
- Transactional memory.
- Streaming architectures, vectorization, and SIMD (single instruction, multiple data hardware units).
- High-performance programming language

About the Course

- Prerequisites

- Remember what we learned about semaphores and mutexes
- Feel comfortable with programming, debugging, etc.
- Feel comfortable with learning a new language (Rust!)

General Information

- Over 400 students
- Three sections
- Pick your preference
- Jeff will do traditional-lecture
- Patrick and I will do some flipped-classrooms

Jeff	08:30-09:50MF	E7 5353
Patrick	11:30-12:50MF	E7 5353
Me	05:30-06:50MW	E7 4053

General Information

- Flipped classrooms means you learn course content and concepts outside the classroom and you do exercises in class
- The in-class exercises are for practice only and not for marks
- But it is a good chance to reinforce your understanding

General Information

- The course has sufficient materials to assist self-learning.
- Lecture materials: https://github.com/jzarnett/ece459
- Video recordings: https://www.youtube.com/playlist?list=PLFCH6yhq9yAGTgG7r30clocD3-QUs9wPL
- The lecture materials are the source materials for video recordings and the content that we say in the class

General Information

- Piazza will be used for questions, discussions, etc.
- You can also email or send messages to me on Teams but it’s always better to post on Piazza since other students might have the same questions

- LEARN will be used for sample exams, grades, tracking grace day usages, etc.

Evaluation

- No midterm exam
- Final exam will be a take-home exam and should be done individually

Academic Integrity Exercise (A Quiz on LEARN)	1%
Assignments	64% (4 at 16% each)
Final exam	35%

Assignments (aka labs)

1. Manual parallelization of a computation using threads; and use of nonblocking I/O
2. Optimizing log file analytics
3. GPU programming with CUDA
4. Open-ended program improvement via profiling

Assignments should be done individually

Assignment hand-in will be done via **git** using the university provided https://git.uwaterloo.ca (GitLab) service

Assignments (aka labs)

- Lab Instructor. The LI is responsible for technical matters related to the assignments: git.uwaterloo.ca, git in general, administering the system where you need to run your labs.
  - **Mike Cooper-Stachowsky – mstachowsky@uwaterloo.ca**

- Teaching Assistants:
  - **Naga Jayanth Chennupati – njchennu@uwaterloo.ca**
  - **Jonathan Chung – jt2chung@uwaterloo.ca**
  - **Tanmayi Jandhyala – tjandhya@uwaterloo.ca**
  - **Vijaya Manikanta Reddy Pothamsetty – m2potham@uwaterloo.ca**
  - **Mohammad Robati Shirzad – mrobatis@uwaterloo.ca**

Final Exam

- It will be a take-home exam
- Open-book, open-notes, but should be done individually
- It will happen during the final exam period
- May contain programming questions

Late policy

- Five grace days
- Grace days are counted in units of whole days
- We look at the last commit time in GitLab on the default branch to determine when your code was submitted.
- No credit for unused late days

Late policy

- Run out of grace days?
- Sixth? lowest assignment mark to be halved.
- Seventh? lowest two assignment marks to be halved.
- More? We'll start converting marks to 0 and dropping the associated late days.
- (Or keep it simple, don't run out of your grace days!)

Late policy

- Some suggestions from the computing education community

- Grace days aren’t excuses for procrastination
- It is found that students who use grace days tend to achieve lower grades than those who don’t
- It may be a trap!

Masks

- Wear a mask in the class for the sake of everyone’s health
- I will wear a mask most of the time, especially when flipped

- The syllabus is also on GitHub
- Check it out if you have further question

Performance!

ECE459: Programming for Performance

What is the meaning of "Performance"?

What is Performance?

- Definitely not mean how to program on stage when other people are watching
- Think about where you use the word “performance”
  - A student gets a full mark in the final exam
  - There is a Hello World program written in C and it runs correctly
  - An AI recognize a person face in a photo of hundred people
  - A search engine can handle million requests per second
  - etc.

What is Performance?

- Probably most of the time, it means

- Correctness
- Efficiency

Performance

- Correctness matters in many cases, e.g., accuracy in machine learning
- But we care more about efficiency in the course

"Many modern software systems must process large amounts of data, either in the form of huge data sets or vast numbers of (concurrent) transactions."

- Note that it does not mean correctness is not important
- It just means we assume correctness is guaranteed at the first place

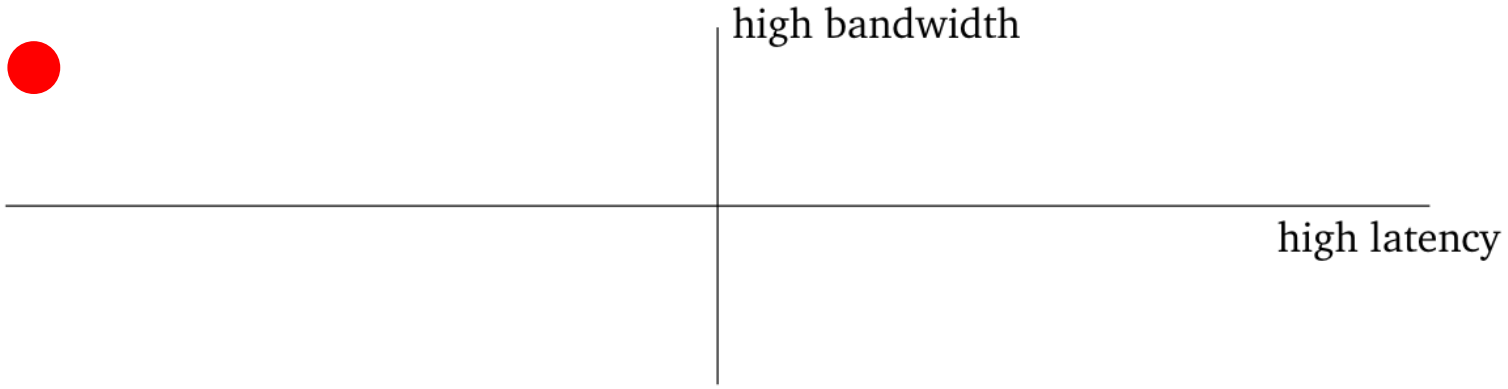
Performance

- Efficiency can be measured by two metrics

- Items per unit time
- Time per item

Performance

- Items per unit time also means bandwidth (or throughput)
- Time per item also means latency
- (What is the best point here?)



Improve performance

- With that said, now we can talk about how to improve the performance
- There are typically two approaches to improve performance

- Improving (reducing) latency
- Do more work at a time

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

```
# Fibonacci numbers
def f(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    return f(n-1) + f(n-2)
```

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

- String-searching problem
- For example, searching a word (W) in a document (S)

```
S:ABC ABCDABAB CDABCDABDE
W:                 ABCDABD
```

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

- Simple solution
  - Two `For` loop

```
S:ABC ABCDABAB CDABCDABDE
W:ABCDABD
W: ABCDABD
W:  ABCDABD
...
```

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

- Simple solution
  - len(W) = k, len(S) = n
  - Worst-case performance is O(k·n)
- Much unnecessary work
  - First three letters are all different

```
S:ABC ABCDABAB CDABCDABDE
W:ABCDABD
W: ABCDABD
W:  ABCDABD
```

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

- Better one
- Knuth–Morris–Pratt algorithm (or KMP algorithm)
- This was the first linear-time algorithm for string matching

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

- KMP algorithm
- Key idea: the word (W) itself embodies sufficient information to determine where the next match could begin, bypassing re-examination of previously matched characters.

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

```
                    1         2
m: 012345678901234567890 12
S: ABC ABCDAB ABCDABCDABDE
W: ABCDABD
i: 0123456
```

https://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

```
                  1         2
m: 012345678901234567890 12
S: ABC ABCDAB ABCDABCDABDE
W:    ABCDABD
i:    0123456
```

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

```
                    1         2
m: 012345678901234567890 12
S: ABC ABCDAB ABCDABCDABDE
W:     ABCDABD
i:     0123456
```

Just prior to the end of the current partial match, there was that substring "AB" that could be the beginning of a new match

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

```
                    1         2
m: 012345678901234567890 12
S: ABC ABCDAB ABCDABCDABDE
W:        ABCDABD
i:        0123456
```

Note that the first "AB" don't need to re-checked
They are guaranteed to match

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

```
                    1         2
m: 012345678901234567890 12
S: ABC ABCDAB ABCDABCDABDE
W:             ABCDABD
i:             0123456
```

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

```
                    1         2
m: 012345678901234567890 12
S: ABC ABCDAB ABCDABCDABDE
W:            ABCDABD
i:            0123456
```

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

```
                    1         2
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W:                ABCDABD
i:                0123456
```

Found
Worst-case performance is O(n) instead of O(k·n)

Improving latency

- Do less work
  - avoid (re)calculating intermediate results

- Producing text output to a log file or to a console screen is expensive
- Only log necessary information
- Bunch of studies are working on how to achieve that

Improving latency

- Do less work
  - avoid (re)calculating intermediate results
  - computing results to only the accuracy that you need in the final output

- Do you always need exact numbers from a large database?
- Or you probably just want numbers with a certain level of accuracy
- Approximate Query Processing (AQP)

Improving latency

- Do less work
  - avoid (re)calculating intermediate results
  - computing results to only the accuracy that you need in the final output

- One idea for AQP
- Data are coming from the real world
- They follow an unknown distribution
- You estimate the distribution and return **approximated aggregates** efficiently

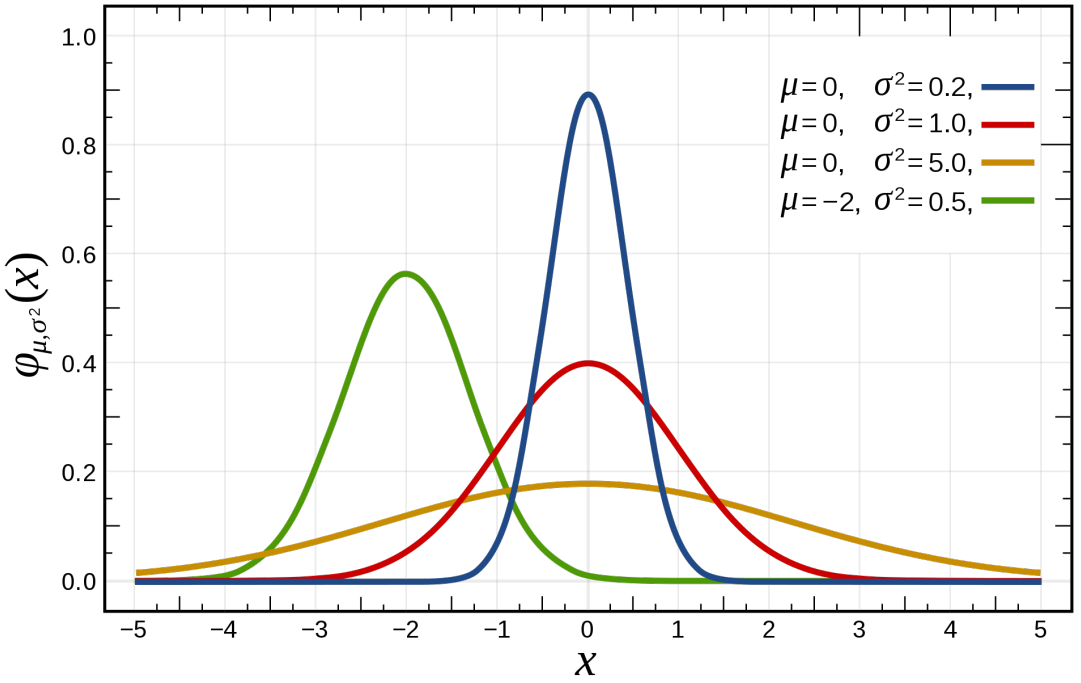
Improving latency

- Do less work
  - avoid (re)calculating intermediate results
  - computing results to only the accuracy that you need in the final output

- The sum of 1, 2, 3, …, 100
- Instead of summing them one by one
- You can calculate as Gauss did
- (1+100)*100/2

Improving latency

- Do less work
  - avoid (re)calculating intermediate results
  - computing results to only the accuracy that you need in the final output



Improving latency

- Do less work
  - avoid (re)calculating intermediate results
  - computing results to only the accuracy that you need in the final output
- Returning exact data is possible if the underlying model is precise, e.g., physics laws
- Sounds like machine learning, huh?

Improving latency

- Do less work
- Be prepared

- Pre-generated reports
  - Request transcript
- Pre-aggregation of Online Analytical Processing (OLAP) data, etc.
  - What's the annual profit of last year?

Improving latency

- Do less work
- Be prepared
- Be smarter

- Better algorithms
  - From bubble sort to quicksort, etc.
- Compiler optimization to get smaller constant factor
- Aware of cache and data locality/density issues

Improving latency

- Do less work
- Be prepared
- Be smarter

- Caching is a hybrid between “do less work” and “be smarter”
- Temporarily store the results
- Caching is really important in certain situations.

Improving latency

- Do less work
- Be prepared
- Be smarter

- Usually done in libraries
- Commonly, you just pick what libraries you want to use, instead of writing it on your own
- Improve once and benefit all

Improving latency

- Do less work
- Be prepared
- Be smarter
- Improve the hardware

- If your computer is too slow, buy a new one …
- Hard disk drive (HDD) => Solid-State Drives (SSD)
- Larger memory
- Better CPU (although it does not necessarily mean higher clock speed)
- Better GPU, etc.

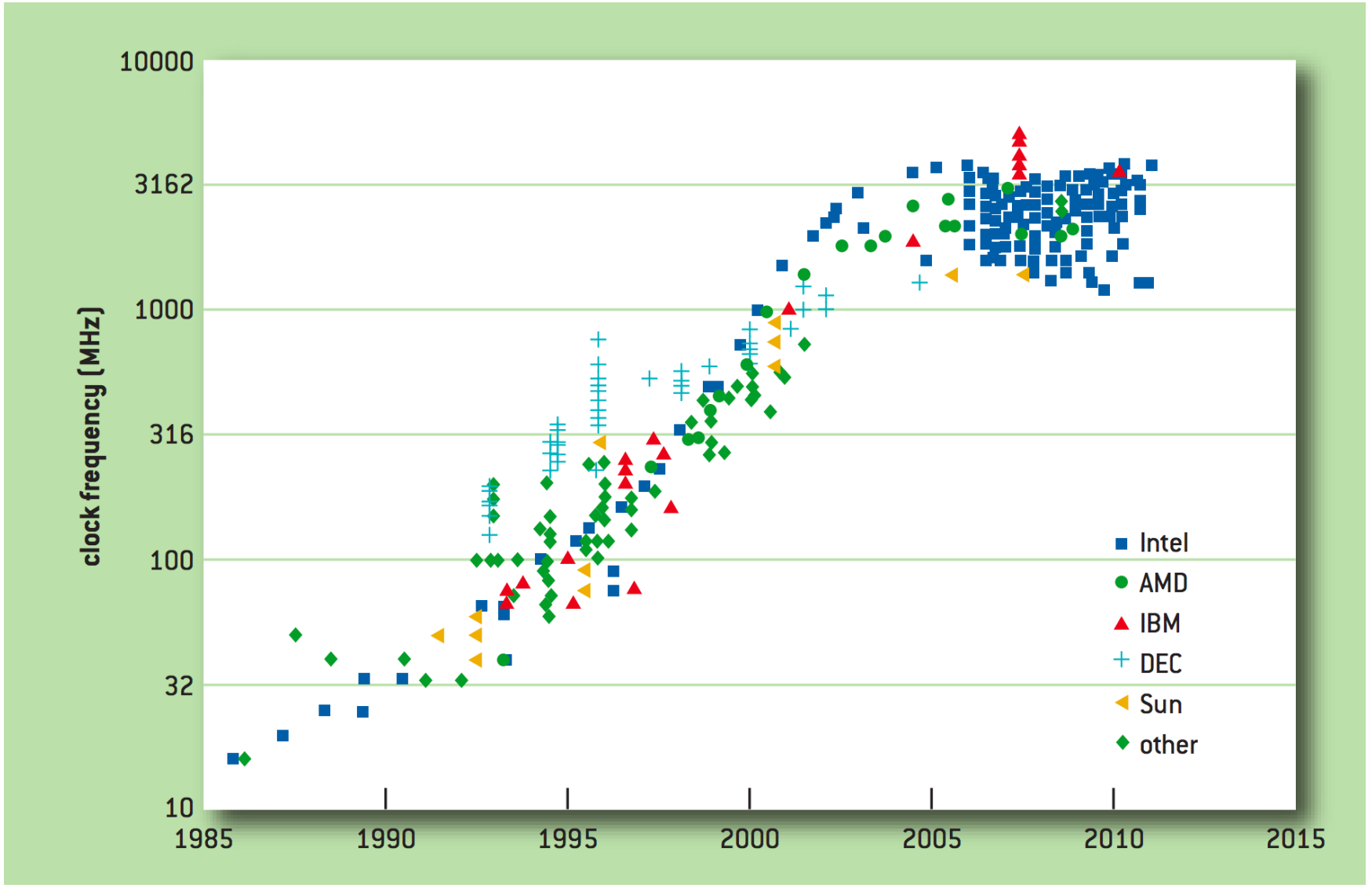
Improving latency

- Do less work
- Be prepared
- Be smarter
- Improve the hardware
- Using assembly code

- Compiler transforms your high-level code to assembly code (kind of like text representation of machine code)
- Compilers may be not very smart

Do more work at a time

- While you may frequently hear about multi-threading, parallel-processing, multi-core CPUs, distributed systems, etc.
- They were not popular phrases 20 years ago.
- Why do we want to look at them these days?



CPU clock speed (frequency) over time [DKM+12]

Do more work at a time

- In general, parallelism improves bandwidth, but not latency.
  - For example, a web server can handle multiple requests simultaneously, but each request still needs the same time to handle.

- However, parallelism also complicates your life

Do more work at a time

- Different problems are amenable to different sorts of parallelization
- Depending on how you look at the problem, you may discover different approaches to parallelize things
  - Algorithm itself?
  - CPU Pipelining?
  - Single Instruction Multiple Data (SIMD)?
  - etc.

Do more work at a time

Clock Cycle	1	2	3	4	5	6	7	8	9	10
	IF 1									
		DC 1								
			OP 1							
				EX 1						
					WB 1					
						IF 2				
							DC 2			
								OP 2		
									EX 2	
										WB 2
	IF 1									
		DC 1								
		IF 2	OP 1							
			DC 2	EX 1						
				OP 2	WB 1					
					EX 2					
						WB 2				

CPU pipelining

Do more work at a time

- Single Instruction Multiple Data (SIMD)

Do more work at a time

- Real parallelism needs hardware supports
- Which can execute multiple instruction streams simultaneously
- Multicore processors, SMP (symmetric multiprocessor) systems, or a cluster of machines
- Or we can put some work on to graphics processing units (GPUs)

Difficulties with using parallelism

- Coordination overhead
  - Sometimes it is easier to do a project when it's just you rather than being you and a team
- Sequential part dominates the program (Amdahl's Law)
  - Trivial improvement from parallelizing other parts
- Ordering
  - Some events A are guaranteed to happen before other events B, but many events X and Y can occur in either the order XY or Y X
  - Data races
  - Deadlock

Exceptions

- Embarrassingly parallel: minimal communication needed between processors

- E.g., Monte Carlo integration

Scalability

- Handle more work with more recourses?
- Even the most scalable systems have their limits

Don’t guess; measure

- It is important to profile you code
- Intuition seems to be often wrong here
- Run your program with realistic workloads
- Checkout http://computers-are-fast.github.io (see lecture 1 notes)

Rust

ECE459: Programming for Performance

Why Rust?

- Remember the time when you struggled with memory leaks and race conditions?
- A lot of the problems we frequently encounter are the kind that can be found by Valgrind, such as memory errors or race conditions.
- However, Valgrind is a runtime checker

Why Rust?

- A design goal of Rust is to avoid issues with memory allocation and concurrency
- Compile-time checking
- Will gradually replace C/C++ to some extent
  - Linux 6.1: Rust to hit mainline kernel
  - https://www.theregister.com/2022/10/05/rust_kernel_pull_request_pulled/

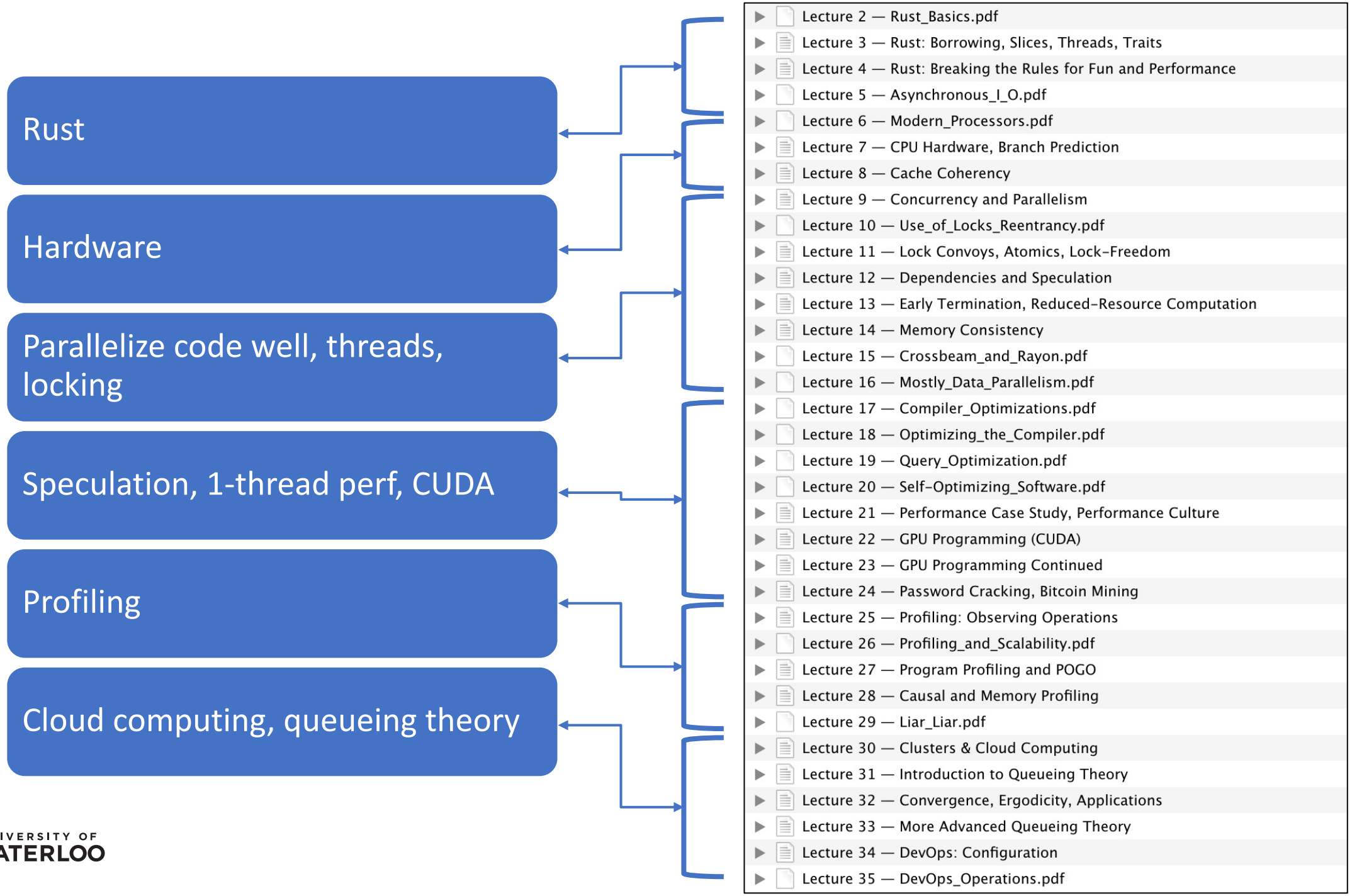
Trade-off in using Rust

- In particular, you will find some things harder to code in Rust than in C/C++
- However, they are also more likely to be correct (remember we care about correctness and assume it at the first place!)
- You put more time in compiler-time debugging instead of in runtime

Trade-off ir

- It may seem lik developers have put a lot c hat try to help.
- As always, plea ues and course staff for help.

https://infosec.exchange/@AstraKernel/109636334567212340


Rust
Hardware
Parallelize code well, threads, locking
Speculation, 1-thread perf, CUDA
Profiling
Cloud computing, queueing theory
Lecture 2 — Rust_Basics.pdf
Lecture 3 — Rust: Borrowing, Slices, Threads, Traits
Lecture 4 — Rust: Breaking the Rules for Fun and Performance
Lecture 5 — Asynchronous_I_O.pdf
Lecture 6 — Modern_Processors.pdf
Lecture 7 — CPU Hardware, Branch Prediction
Lecture 8 — Cache Coherency
Lecture 9 — Concurrency and Parallelism
Lecture 10 — Use_of_Locks_Reentrancy.pdf
Lecture 11 — Lock Convoys, Atomics, Lock-Freedom
Lecture 12 — Dependencies and Speculation
Lecture 13 — Early Termination, Reduced-Resource Computation
Lecture 14 — Memory Consistency
Lecture 15 — Crossbeam_and_Rayon.pdf
Lecture 16 — Mostly_Data_Parallelism.pdf
Lecture 17 — Compiler_Optimizations.pdf
Lecture 18 — Optimizing_the_Compiler.pdf
Lecture 19 — Query_Optimization.pdf
Lecture 20 — Self-Optimizing_Software.pdf
Lecture 21 — Performance Case Study, Performance Culture
Lecture 22 — GPU Programming (CUDA)
Lecture 23 — GPU Programming Continued
Lecture 24 — Password Cracking, Bitcoin Mining
Lecture 25 — Profiling: Observing Operations
Lecture 26 — Profiling_and_Scalability.pdf
Lecture 27 — Program Profiling and POGO
Lecture 28 — Causal and Memory Profiling
Lecture 29 — Liar_Liar.pdf
Lecture 30 — Clusters & Cloud Computing
Lecture 31 — Introduction to Queueing Theory
Lecture 32 — Convergence, Ergodicity, Applications
Lecture 33 — More Advanced Queueing Theory
Lecture 34 — DevOps: Configuration
Lecture 35 — DevOps_Operations.pdf

Prepare for the next lecture

- **Setup GitLab if you haven't done so (!)**
- Read the lecture 2 notes
- Read relevant chapters of the Rust book (https://doc.rust-lang.org/stable/book/)
- Setup Rust development environment
- Bring your laptops and we will do some in-class exercises