Introduction I

Overview

- Today we will discuss the course content, structure and marking scheme
- We will begin to learn about **multicore** and **many-core** programming
- We will also consider the relationship between computer hardware and programming

CSCI 4060U

Massively Parallel Programming

- **Instructor:** Dr. Jeremy S. Bradbury

  Office hours:

  - Tuesdays 1:00pm-2:00pm (in-person),
  - Fridays 10:00am-11:00am (virtual),
  - otherwise by appointment
- **Teaching Assistant:** Andrei Stoica

  Office hours:

  - during labs, otherwise by appointment

CSCI 4060U

Massively Parallel Programming

- **Lectures:**
  - Tuesday 8:10am-9:30am
  - Thursday 8:10am-9:30am

- **Laboratories:**
  - Tuesday 9:40am-11:00am
  - Wednesday 3:40pm-5:00pm

Labs start the week of Jan. 20, 2025

CSCI 4060U

Massively Parallel Programming

- No required textbook
- We will be using online resources
  - http://www.sqrlab.ca/csci4060u/resources-links/
    (updated throughout the semester)

CSCI 4060U

Massively Parallel Programming

- **Learning Outcomes:**
  - Understand the challenges of programming with multicore, many-core and massively parallel computer systems
  - Develop applied knowledge of multicore programming approaches, strategies and design patterns
  - Develop applied knowledge of many-core programming approaches, strategies and design patterns
  - Understand how to debug multicore and many-core source code

CSCI 4060U

Massively Parallel Programming

- Topics:
  - **Introduction** (1 week)
    - Overview of shared memory vs. distributed memory processing
    - Overview of CPU & GPU hardware
    - Overview of the challenges of parallel computation
    - Approaches to parallel programming – preprocessor directives, threads, actors

CSCI 4060U

Massively Parallel Programming

- Topics:
  - **OpenMP Programming** (3 weeks)
    - Introduction to preprocessor directives and OpenMP (in C++)
    - OpenMP Memory Model
    - Sharing work between threads (e.g., loop, sections and workshare constructs)
    - Controlling work-sharing constructs (e.g., shared and private clauses)
    - Synchronization (e.g., atomic construct, locks)

CSCI 4060U

Massively Parallel Programming

- Topics:
  - **Thread Programming** (3 weeks)
    - Explicit vs implicit threading
    - Introduction to programming with threads (C++ POSIX threads)
    - Managing threads (e.g., creation)
    - Mutex variables (locking, unlocking)
    - Conditional variables
    - Debugging

POSIX Threads

CSCI 4060U

Massively Parallel Programming

- Topics:
  - **OpenCL Programming** (1 week)
    - Introduction to task and data parallelism in OpenCL
    - OpenCL Programming Model
    - The anatomy of an OpenCL program (kernel, host)
    - Working with data in OpenCL – dividing up your data, sending data to and getting results from the kernel
    - Debugging

CSCI 4060U

Massively Parallel Programming

- Topics:
  - **Applications of Massively Parallel Programming** (1.5 week)
    - *Heterogeneous Computing* (0.5 weeks): Looking at the future of multicore and many-core programming and exploring how we can leverage both the CPU and GPU together.
    - Exploring the importance of *massively parallel programming to AI* (1 week)

CSCI 4060U

Massively Parallel Programming

- Marking:

Tests (3)	**40%**
Laboratories (8)	**40%**
Final Project	**20%**

CSCI 4060U

Massively Parallel Programming

- Tests:
  - Test #1 – Tuesday, Feb. 4, 2025 (in-class)
    - Introductory content, OpenMP
  - Test #2 – Thursday, Mar. 6, 2025 (in-class)
    - Threads
  - Test #3 – Thursday, Apr. 3, 2025 (take home)
    - OpenCL, heterogeneous computing, parallel programming & AI

CSCI 4060U

Massively Parallel Programming

- Marking:

Tests (3)	**40%**
Laboratories (8)	**40%**
Final Project	**20%**

CSCI 4060U

Massively Parallel Programming

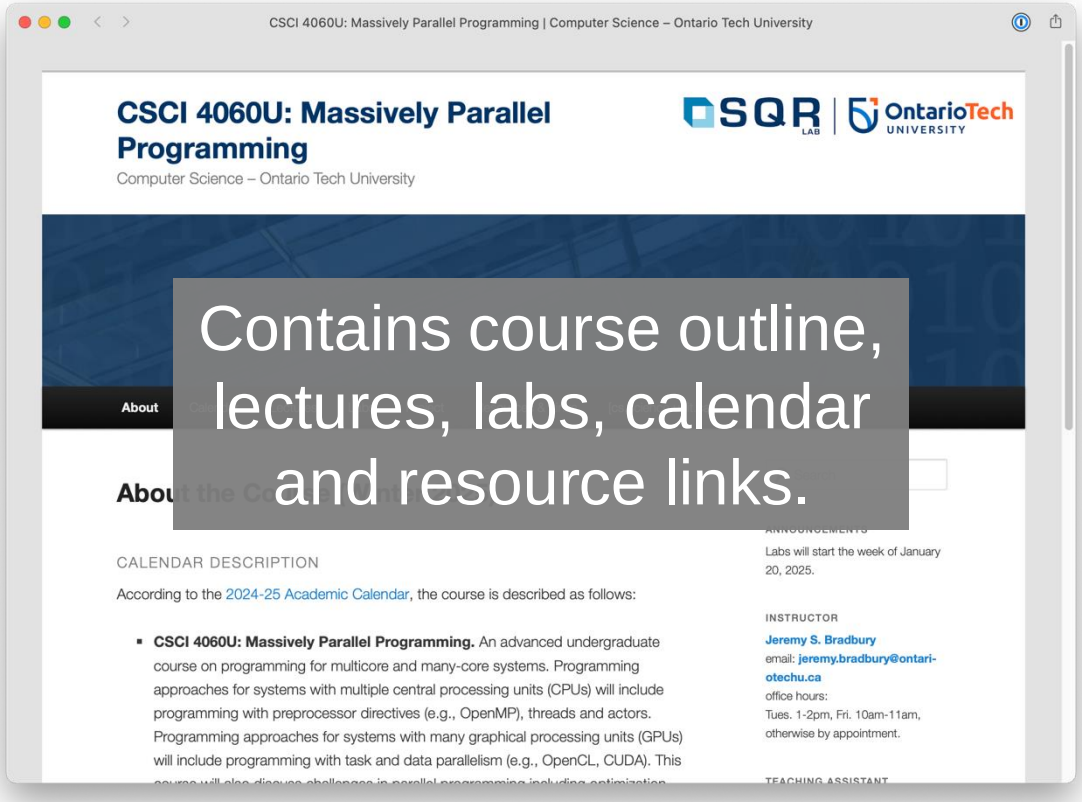
- Project:
  - *Concurrency Paper:* Write a paper providing an overview of concurrency in a language not covered during the lectures **OR**
  - *Concurrent Program:* Create a concurrent program that demonstrated your understanding of the concurrency concepts discussed in class.
  - Deliverables:
    - Proposal – Friday, Feb. 28, 2025
    - Final Submission – Monday, Apr. 14, 2025
    - Presentation (pre-recorded) – Monday, Apr. 14, 2025

More Information?

- Course website: http://www.sqrlab.ca/csci4060u/

More Information?

- Course website: http://www.sqrlab.ca/csci4060u/



Contacting Your Professor/TA

- Slack: http://csci4060u-w25.slack.com/

Contacting Your Professor/TA

- Slack: http://csci4060u-w25.slack.com/

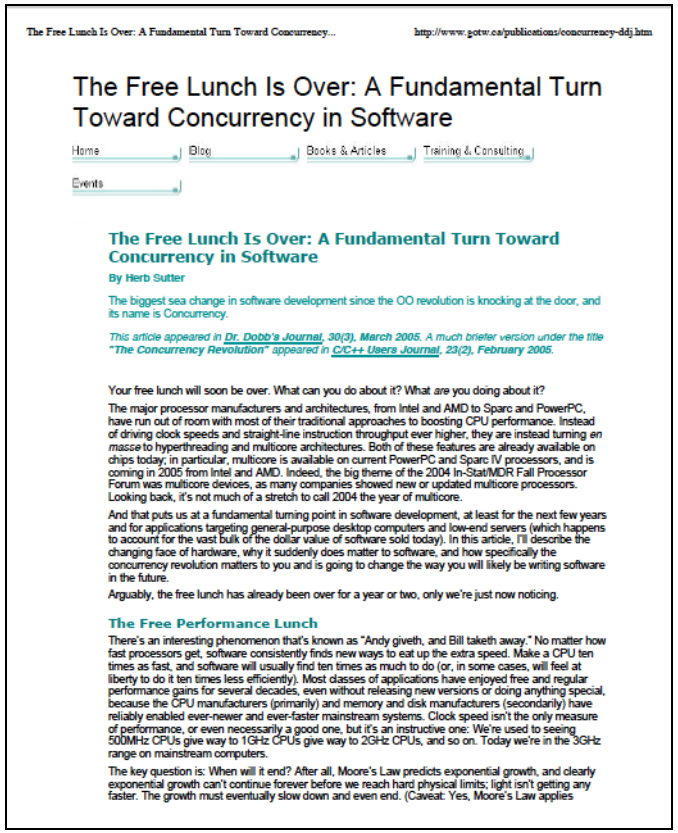
Used for discussions, questions & answers and course announcements. If you need to contact your professor or TA – try here first!

What is Concurrency?

> **"Concurrency occurs when two or more execution flows (threads) are able to run simultaneously."**
>
> **- Edsger Dijkstra**

Software's Free Lunch – and why it is over

> **"The good news is that processors are going to continue to become more powerful. The bad news is that, at least in the short term, the growth will come mostly in directions that do not take most current applications along for their customary free ride."**
>
> **- Herb Sutter [Sut05]**



The Free Lunch Is Over: A Fundamental Turn Toward Concurrency... http://www.gotw.ca/publications/concurrency-ddj.htm

The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software

Home | Blog | Books & Articles | Training & Consulting | Events

The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software

By Herb Sutter

The biggest sea change in software development since the OO revolution is knocking at the door, and its name is Concurrency.

This article appeared in Dr. Dobb's Journal, 30(3), March 2005. A much briefer version under the title "The Concurrency Revolution" appeared in C/C++ Users Journal, 23(2), February 2005.

Your free lunch will soon be over. What can you do about it? What *are* you doing about it?

The major processor manufacturers and architectures, from Intel and AMD to Sparc and PowerPC, have run out of room with most of their traditional approaches to boosting CPU performance. Instead of driving clock speeds and straight-line instruction throughput ever higher, they are instead turning *en masse* to hyperthreading and multicore architectures. Both of these features are already available on chips today; in particular, multicore is available on current PowerPC and Sparc IV processors, and is coming in 2005 from Intel and AMD. Indeed, the big theme of the 2004 In-Stat/MDR Fall Processor Forum was multicore devices, as many companies showed new or updated multicore processors. Looking back, it's not much of a stretch to call 2004 the year of multicore.

And that puts us at a fundamental turning point in software development, at least for the next few years and for applications targeting general-purpose desktop computers and low-end servers (which happens to account for the vast bulk of the dollar value of software sold today). In this article, I'll describe the changing face of hardware, why it suddenly does matter to software, and how specifically the concurrency revolution matters to you and is going to change the way you will likely be writing software in the future.

Arguably, the free lunch has already been over for a year or two, only we're just now noticing.

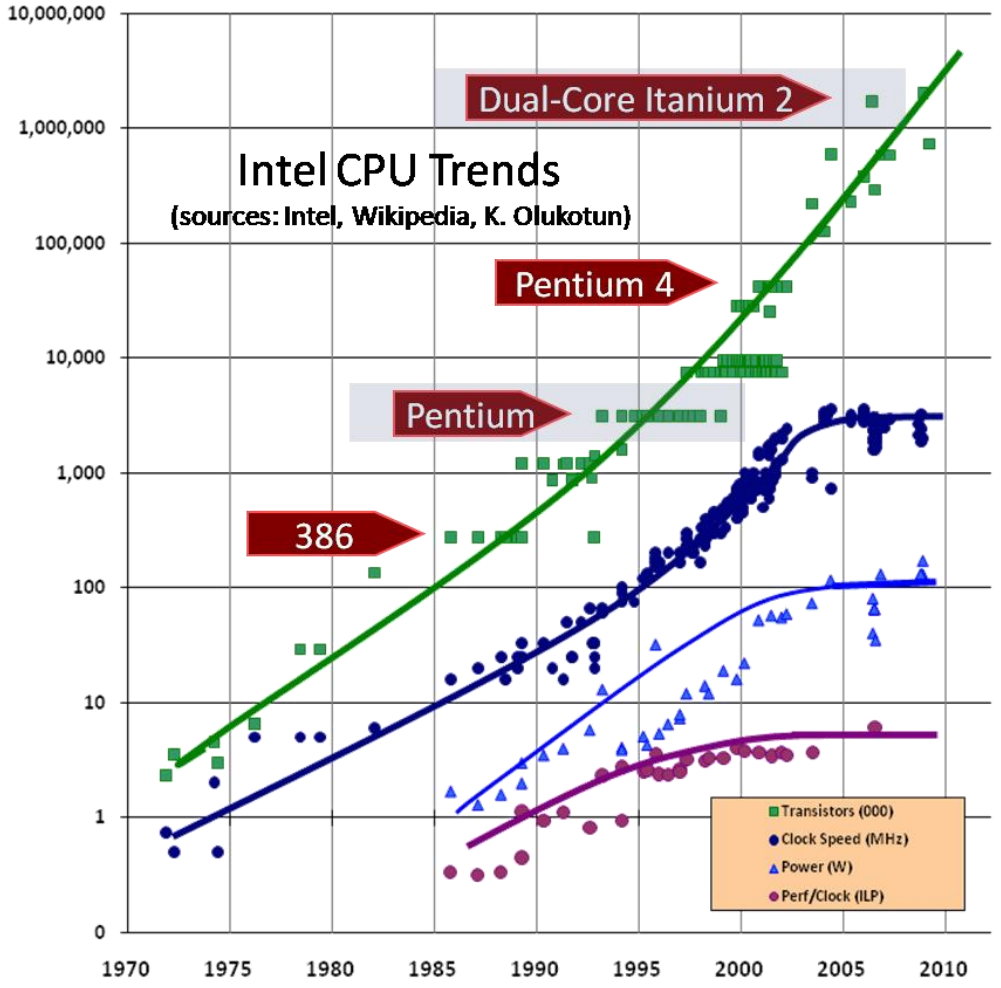
The Free Performance Lunch

There's an interesting phenomenon that's known as "Andy giveth, and Bill taketh away." No matter how fast processors get, software consistently finds new ways to eat up the extra speed. Make a CPU ten times as fast, and software will usually find ten times as much to do (or, in some cases, will feel at liberty to do it ten times less efficiently). Most classes of applications have enjoyed free and regular performance gains for several decades, even without releasing new versions or doing anything special, because the CPU manufacturers (primarily) and memory and disk manufacturers (secondarily) have reliably enabled ever-newer and ever-faster mainstream systems. Clock speed isn't the only measure of performance, or even necessarily a good one, but it's an instructive one: We're used to seeing 500MHz CPUs give way to 1GHz CPUs give way to 2GHz CPUs, and so on. Today we're in the 3GHz range on mainstream computers.

The key question is: When will it end? After all, Moore's Law predicts exponential growth, and clearly exponential growth can't continue forever before we reach hard physical limits; light isn't getting any faster. The growth must eventually slow down and even end. (Caveat: Yes, Moore's Law applies

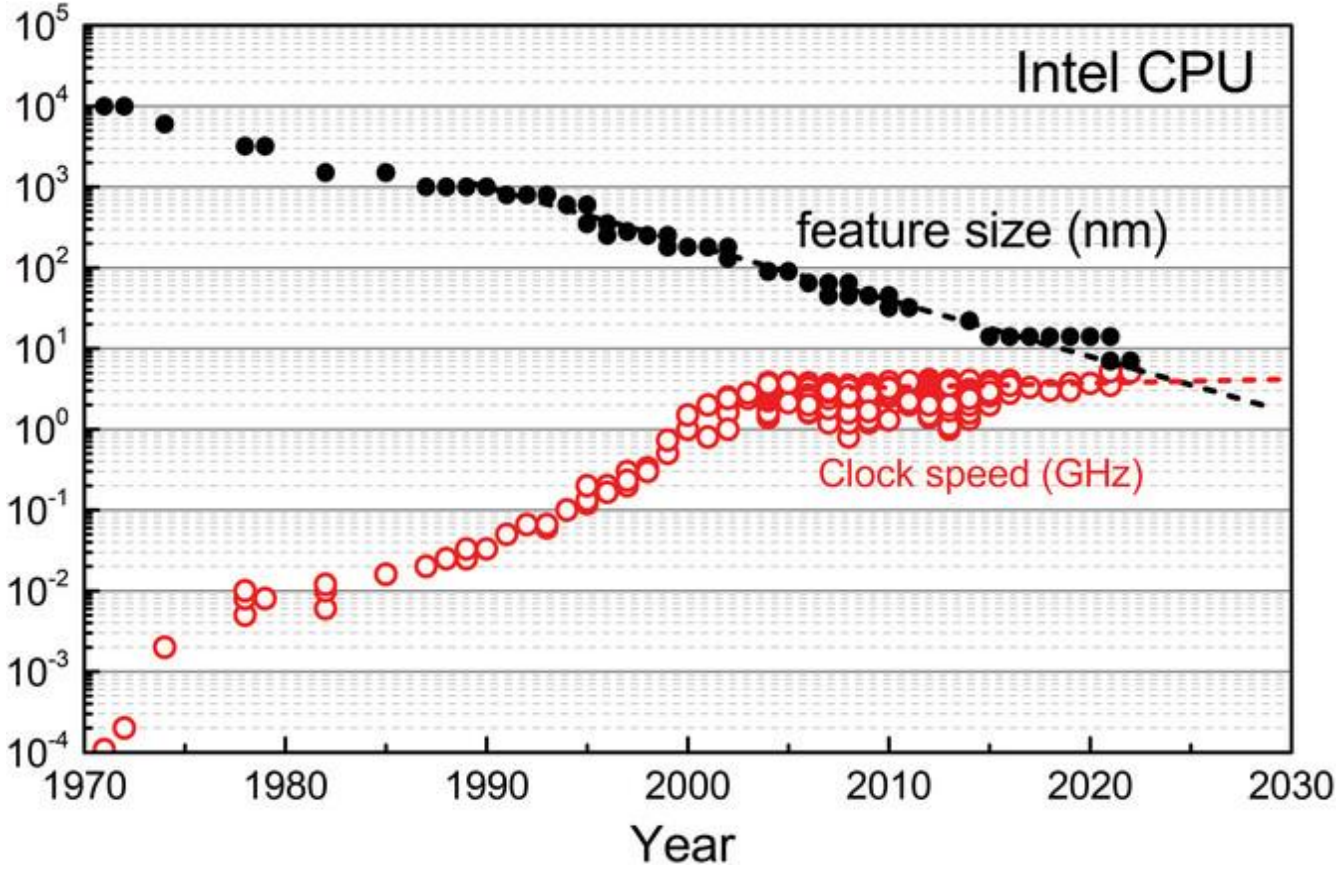
[Sut05] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), Mar. 2005.

Software's Free Lunch – and why it is over

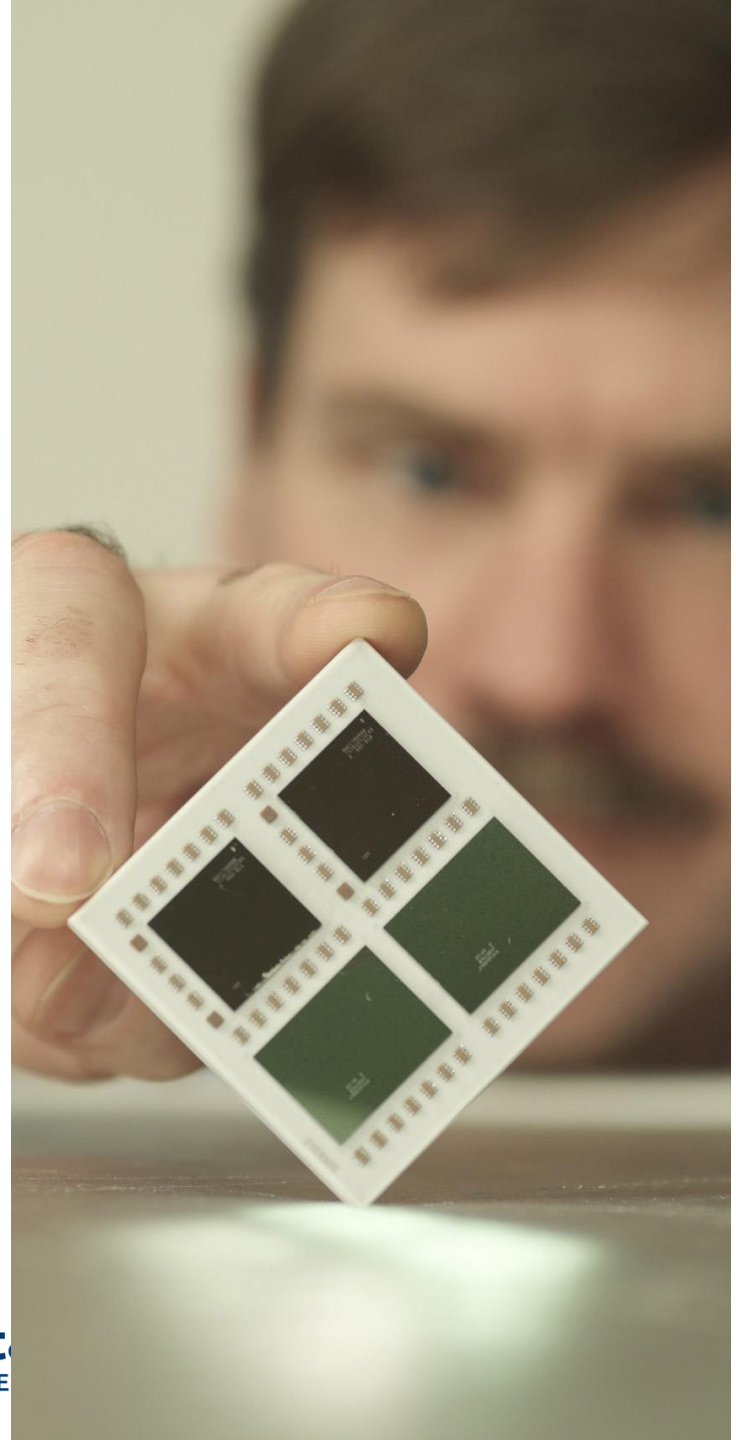


source:
http://www.gotw.ca/publications/concurrency-ddj.htm

Software's Free Lunch – and why it is over



source: L. Zhu, Switching of Perpendicular Magnetization by Spin–Orbit Torque. *Adv. Mater.* 2023, 35, 2300853. https://doi.org/10.1002/adma.202300853



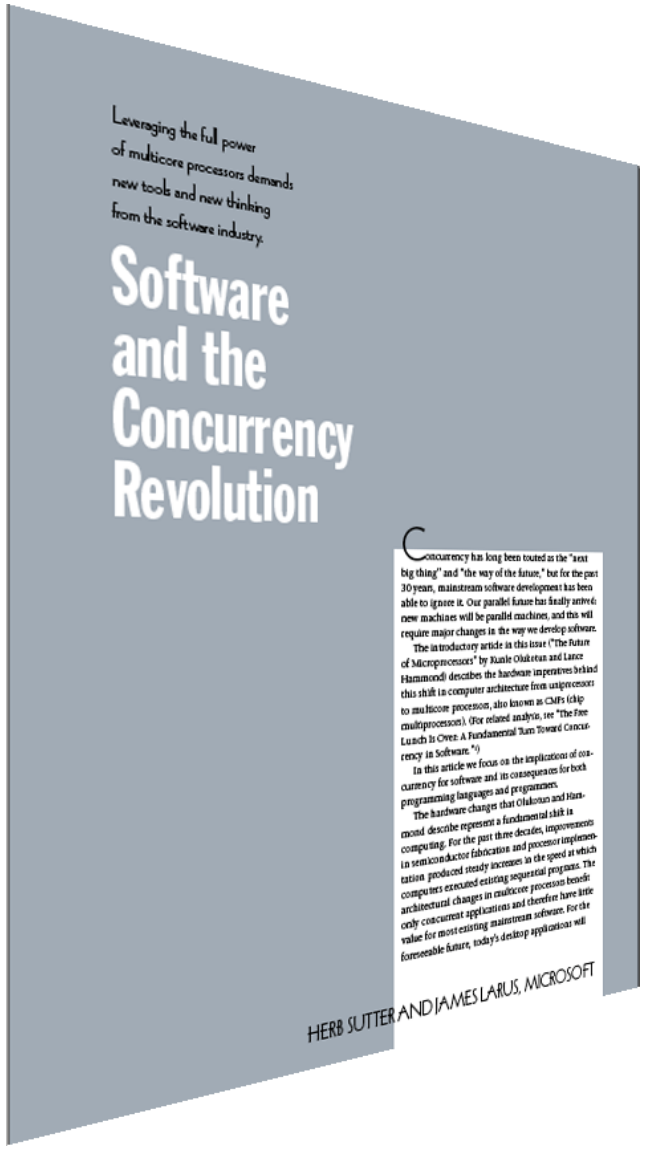
In the future applications will need to be **concurrent** to fully exploit CPU throughput gains [Sut05]

[Sut05] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), Mar. 2005.

Why is Concurrency Difficult?

1. The many different, possibly unexpected, executions of the program

2. The sharing of data and resources between threads



> **"...humans are quickly overwhelmed by concurrency and find it much more difficult to reason about concurrent than sequential code. Even careful people miss possible interleavings..."**

- Herb Sutter & James Larus, Microsoft [SL05]

[SL05] H. Sutter and J. Larus. Software and the concurrency revolution. Queue, 3(7):54–62, 2005.

> I conjecture that most multithreaded general purpose application are so full of concurrency bugs that - as multicore architectures become commonplace - these bugs will begin to show up as system failures.

- Edward A. Lee [Lee06]

COVER FEATURE

Future Direction

The Problem with Threads

Edward A. Lee
University of California, Berkeley

For concurrent programming to become mainstream, we must discard threads as a programming model. Nondeterminism should be judiciously and carefully introduced where needed, and it should be explicit in programs.

Concurrent programming is difficult,[1] yet many technologists predict the end of Moore's law will be answered with increasingly parallel computer architectures—multicore or chip multiprocessors (CMPs).[2] If we hope to achieve continued performance gains, programs must be able to exploit this parallelism.

Automatic exploitation of parallelism in sequential programs, through either computer architecture techniques such as dynamic dispatch or automatic parallelization of sequential programs,[3] offers one possible technical solution. However, many researchers agree that these automatic techniques have been pushed to their limits and can exploit only modest parallelism. Thus, programs themselves must become more concurrent.

Understanding why concurrent programming is so difficult can help us solve the problem. The physical world is highly concurrent, and our very survival depends on our ability to reason about concurrent physical dynamics. This reasoning doesn't extend to concurrent programs because we have chosen abstractions that do not even vaguely resemble the physical world's concurrency. We have become so used to these computational abstractions that we have forgotten they are not immutable. The difficulty of concurrent programming is a consequence of these abstractions, and if we can let go of them, the problem will be tirable.

THREADS

In general-purpose software engineering practice, we have reached a point where one approach to concurrent programming dominates all others—namely, threads, sequential processes that share memory. They represent a key concurrency model supported by modern computers, programming languages, and operating systems. Many general-purpose parallel architectures in use today—such as symmetric multiprocessors—are direct hardware realizations of the thread abstraction.

Some applications can use threads very effectively—for example, so-called embarrassingly parallel applications that essentially spawn multiple independent processes such as build tools (PVM gmake) or Web servers. Given these applications' independence, programming is relatively easy and the abstraction being used is more like processes than threads. Where such applications do share data, they do so through database abstractions, which manage concurrency through such mechanisms as transactions. However, client-side applications are not so simple.

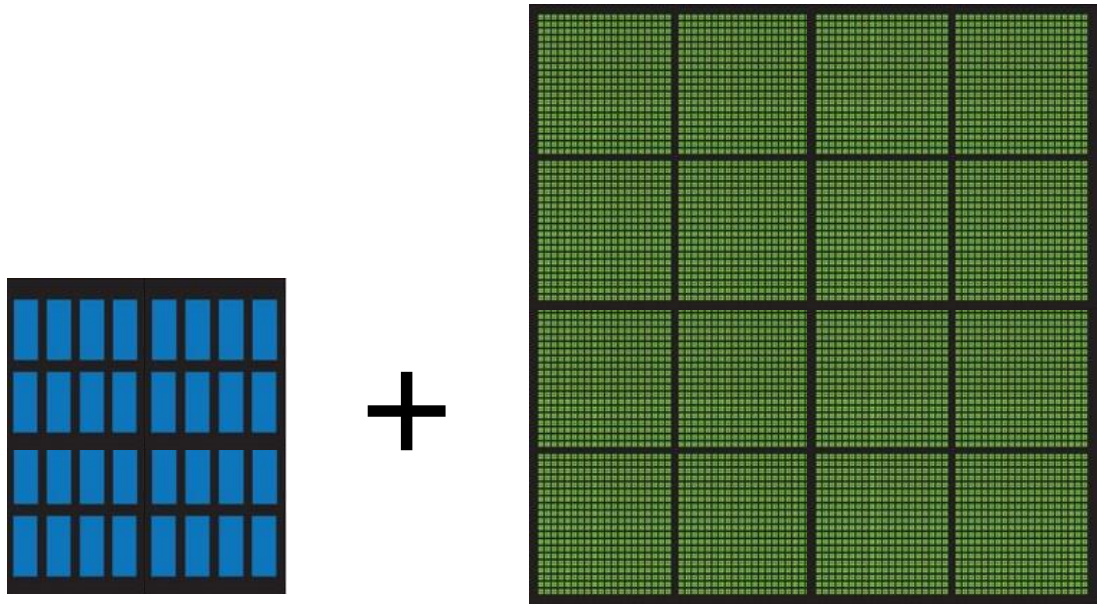
Threads are not the only possibility for concurrent programming. In scientific computing, where performance requirements have long demanded concurrent programming, data-parallel language extensions and message-passing libraries—such as PVM, MPI, and OpenMP—dominate over threads for concurrent programming. Computer architectures intended for scientific computing often differ significantly from so-called general-purpose architectures. They commonly support vectors and streams in hardware, for example. However, even in this domain, concurrent programs remain tedious to write. C and Fortran dominate, despite a long history of much better data-parallel languages.

Published by the IEEE Computer Society

[Lee06] E.A. Lee. The problem with threads. Computer, 39(5):33– 42, May 2006.

What is Massively Parallel?

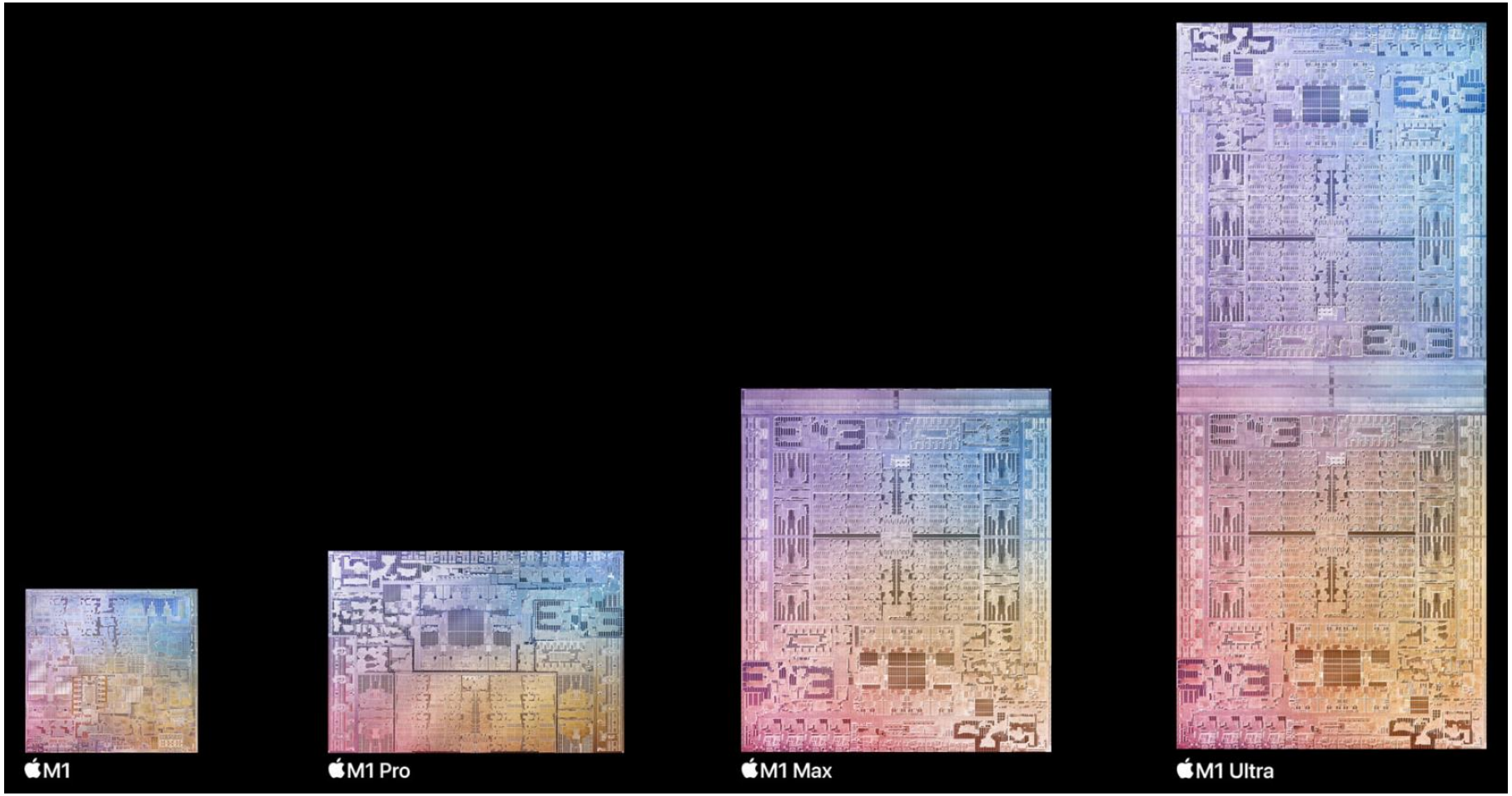
- *"Large number of computer processors"*
- For example:



CPU multicore GPU manycore

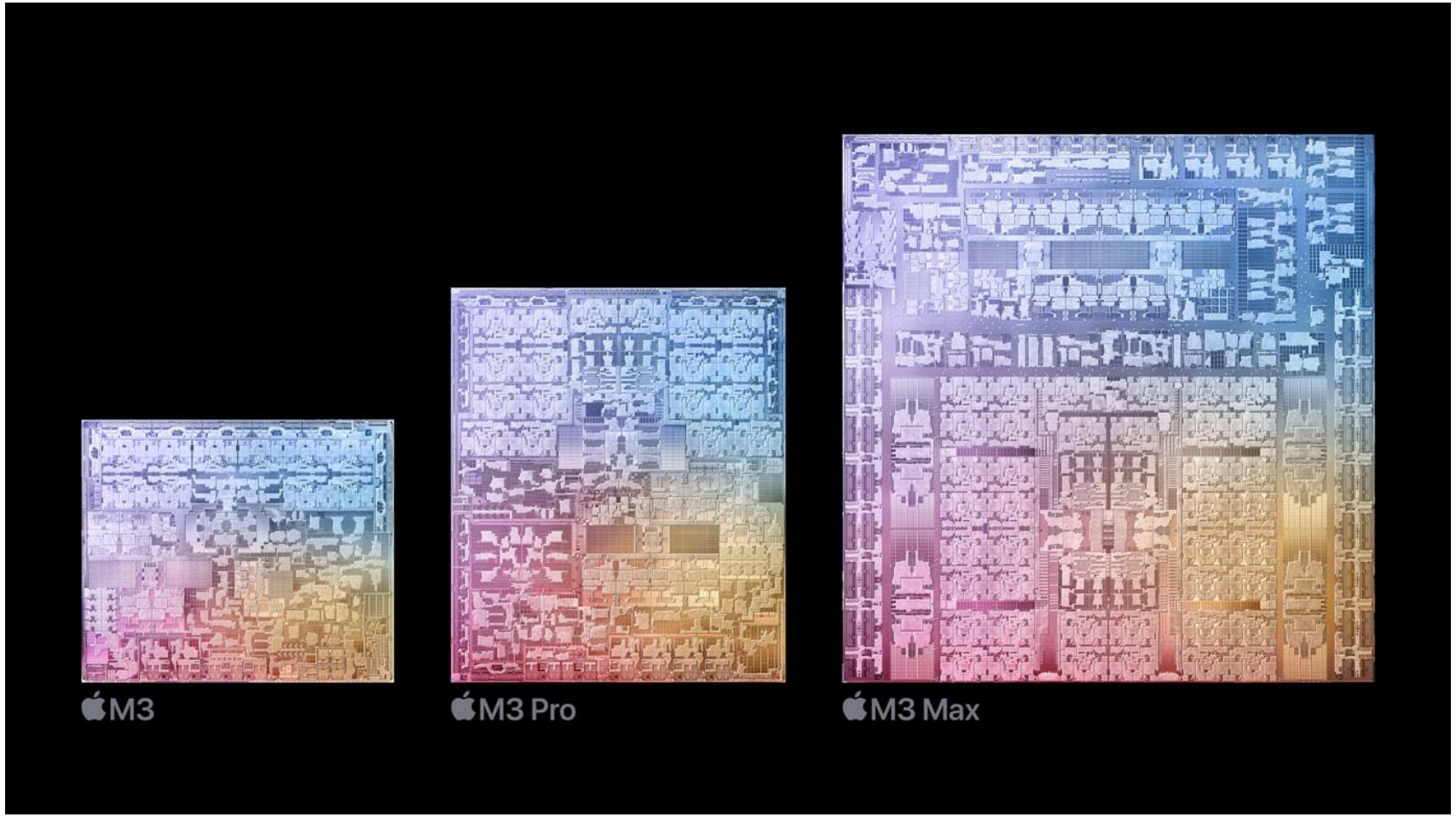
Source: http://www.nvidia.com/docs/IO/143716/cpu-and-gpu.jpg

Case Study: Apple Chips



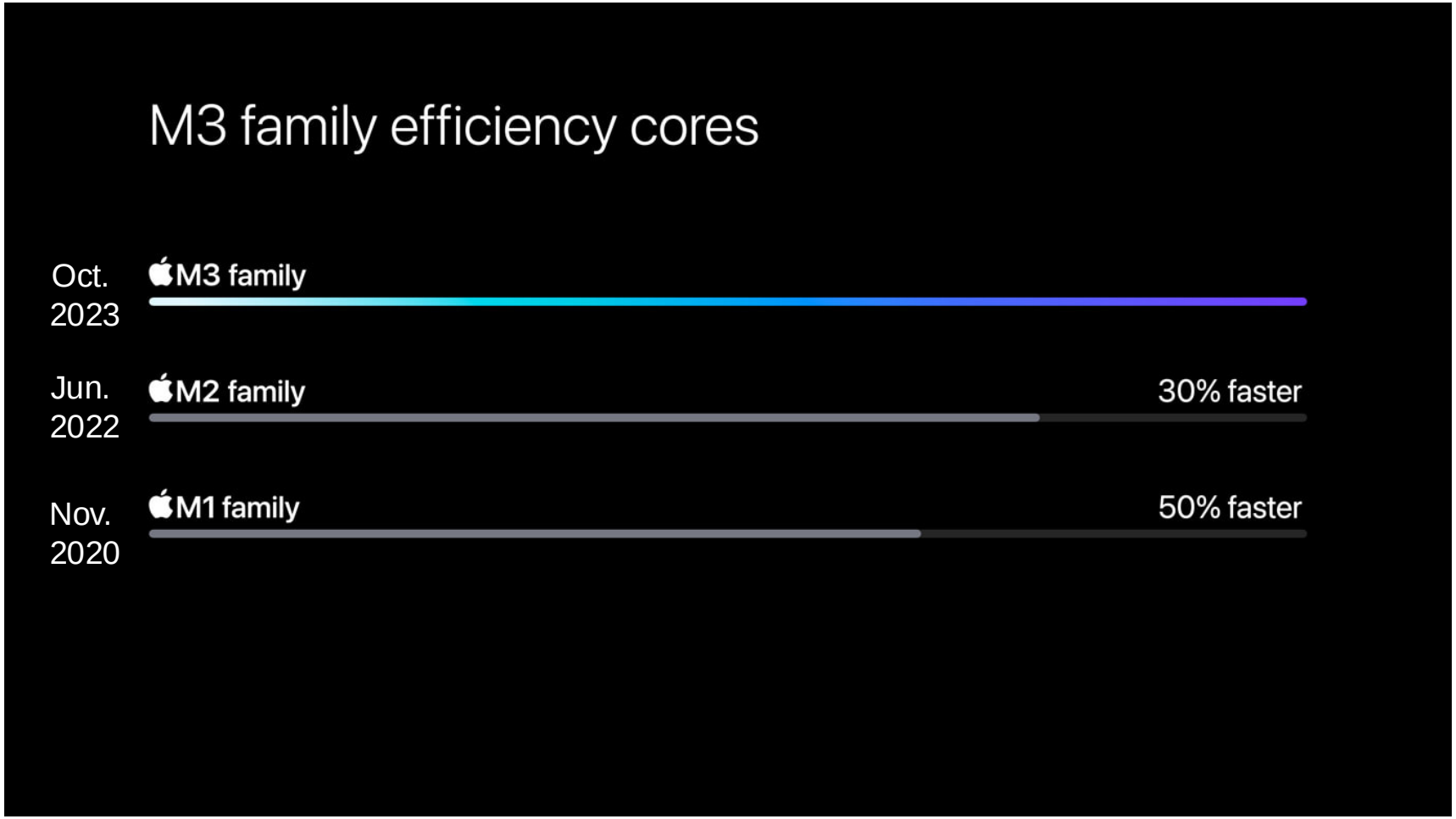
Source: https://www.apple.com/ca/newsroom/2022/03/apple-unveils-m1-ultra-the-worlds-most-powerful-chip-for-a-personal-computer/

Case Study: Apple Chips



Source: https://www.apple.com/newsroom/2023/10/apple-unveils-m3-m3-pro-and-m3-max-the-most-advanced-chips-for-a-personal-computer/

Case Study: Apple Chips



Source: https://www.apple.com/newsroom/2023/10/apple-unveils-m3-m3-pro-and-m3-max-the-most-advanced-chips-for-a-personal-computer/

Introduction I

Summary

- We reviewed the course outline
- We introduced concurrency and the challenges of concurrent programming

Readings

- What's the Difference Between a CPU and a GPU?
- What Makes Parallel Programming Hard?
- The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software
- Apple unveils M1 Ultra, the world's most powerful chip for a personal computer
- Apple unveils M3, M3 Pro, and M3 Max, the most advanced chips for a personal computer

Next time

- A parallel architecture taxonomy – data-level vs. thread-level parallelism