

# Module 1

## Course Logistics & Introduction to Parallel Programming

---

MPCS 52060: Parallel Programming

University of Chicago

# Agenda

1. Course Logistics
2. Introduction to Parallel Programming
  - Motivation for parallelism
  - Evolution of modern machines
  - Why parallelism is important?
3. Course language [if time permits]

- All course information is located on the course website:  
*<https://classes.cs.uchicago.edu/archive/2022/spring/52060-1/index.html>*

## Motivation for Parallelism

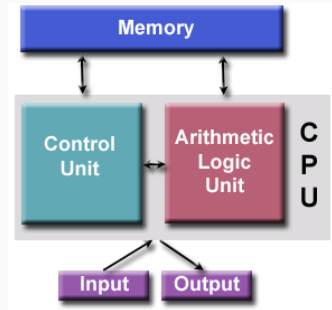
---

# The Basic Architecture of a Computer

Named after the Hungarian mathematician/genius John Von Neumann who first authored the general requirements for an electronic computer in his 1945 papers

He came up with the basic architecture that virtually all computers today still follow (to a degree).

- The architecture is comprised of four main components:
  - Main Memory
  - Control Unit
  - Arithmetic Logic Unit
  - Input/Output



# The Basic Architecture of a Computer (cont.)

- Main Memory
  - This is a collection of locations, each of which is capable of storing both instructions and data.
  - Every location consists of an address, which is used to access the location, and the contents of the location.
- CPU : Divided into two parts
  - **Control unit** - responsible for deciding which instruction in a program should be executed. (the boss)
  - **Arithmetic and logic unit (ALU)** - responsible for executing the actual instructions. (the worker)
- Input/Output
  - is the interface to the human operator

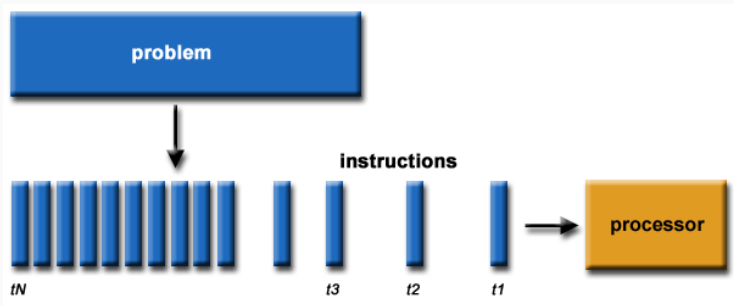
## Why does this all matter?

- Computers still follow this basic design, just multiplied in units (multiprocessor). The fundamental architecture remains the same.

# Serial Programming

As programmers, we are overwhelmingly accustomed to developing software for **serial computation**<sup>1</sup>

- A problem is broken down into discrete steps
- Each step (i.e., instruction) is executed one by one
- Each instruction is executed on a single process
- Only a single instruction is executed at a time



<sup>1</sup>Source: Blaise Barney, Lawrence Livermore National Laboratory

# Serial Programming Example

Simple example of serial computation:

- **Task:** Develop a program that gathers data about route information from different cities. The overall objective is to use this program to help develop a classifier to tell a user whether to buy a ticket now or later.

The screenshot shows the Kayak flight search results for a round-trip from Chicago (CHI) to Atlanta (ATL) on Wed 8/19 to Wed 8/26. A blue box highlights the 'OUR ADVICE Buy now' section, which states: 'Prices are unlikely to decrease within 7 days' and includes a 'Track prices' toggle set to 'OFF'. Below this, it says '4585 of 4599 flights'. The search results are sorted by 'Cheapest' (\$51, 2h 13m). The flight details for the cheapest option are: Spirit Airlines, nonstop, 6:16 pm - 9:29 pm, 2h 13m, ORD - ATL, \$51. Another option is shown: Spirit Airlines, nonstop, 7:03 am - 8:14 am, 2h 11m, ATL - ORD. The interface also includes a 'Flexible options' section with checkboxes for 'Flexible changes' and 'Flexible cancellation'.


**KAYAK** From Booking.com

Flights Hotels Cars Packages Explore Guides More ▾

Round-trip ▾ 2 Travelers ▾ Economy ▾ 0 Bags ▾

Chicago (CHI) ✕ ↔ Atlanta (ATL) ✕


Wed 8/19 < > Wed 8/26 < > 🔍

**OUR ADVICE**  
**Buy now**   
Prices are unlikely to decrease within 7 days ⓘ  
Track prices ☐ OFF


4585 of 4599 flights

We've tagged certain airlines that may be waiving their change or cancellation fees due to COVID-19. Confirm terms and policies on their site. [Learn more](#)

**Cheapest** \$51 • 2h 13m **Best** \$67 • 1h 52m **Quickest** Info • 1h 47m **Other Sort**

**Cheapest** Rating: 9 

<input type="checkbox"/> <b>spirit</b>	<b>6:16 pm – 9:29 pm</b> Spirit Airlines	nonstop	2h 13m ORD - ATL	<b>\$51</b> FlyFar <b>View Deal</b> ▾
<input type="checkbox"/> <b>spirit</b>	<b>7:03 am – 8:14 am</b> Spirit Airlines	nonstop	2h 11m ATL - ORD	

**Flexible options** 

☐ Flexible changes  
☐ Flexible cancellation



# Crawler Components

Components of the application:

- A web crawler that scrapes the *<https://www.faredetective.com/farehistory/>* for route information. Scrapes all cities provided on the website lettered A-Z on different pages.
- **Program Output:** A CSV file with lowest-cost fare information from the cities listed on the site.

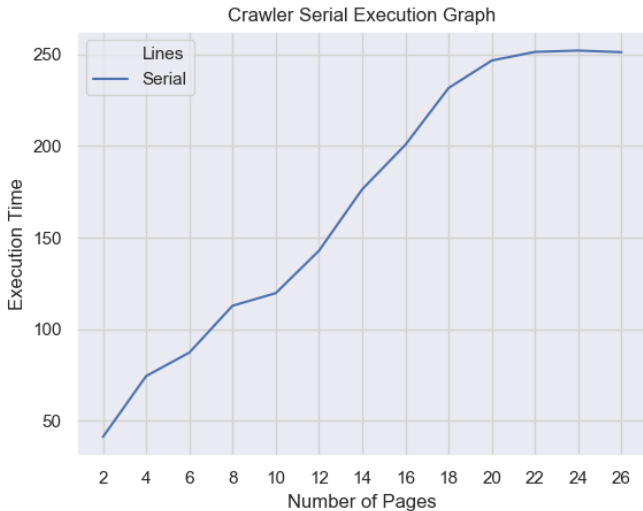
# Crawler Output

Visual of program output:

routes		
From	To	Fare
Amsterdam	Dublin	50
Amsterdam	South Hampton	88
Amsterdam	Birmingham	108
Amsterdam	Belfast	128
Amsterdam	Belfast	136
Amsterdam	Aberdeen	137
Amsterdam	Manchester	185
Amsterdam	Venice	190
Amsterdam	Ljubliana	206
Amsterdam	Bristol	209
Amsterdam	Trieste	236
Amsterdam	Gothenburg	296
Amsterdam	Minneapolis	338
Amsterdam	Boston	391

# Crawler Performance

Execution time (in seconds):



# Is Serial Computation Good Enough?

Since 2002, single-processor performance improvement has slowed to about 20% per year. **Maybe not?**

## History of Hardware Trends

- Increase in single processor performance has been driven by increasing the **density of transistors**.
- *Transistors*: electronic components on integrated circuits that act as switches in order to construct logical gates. We use logic gates to form logical units capable of arithmetic and complex logical operations.
- As the size of the transistors decreases, their speed can be increased, and the overall speed of the integrated chip will be increased (i.e., increasing clock rate).

# Moore's Law

**Moore's Law:** computing power tends to approximately double every two years.

- What the law really means is that the number of transistors that can be packed into a given unit of space will roughly double every two years.

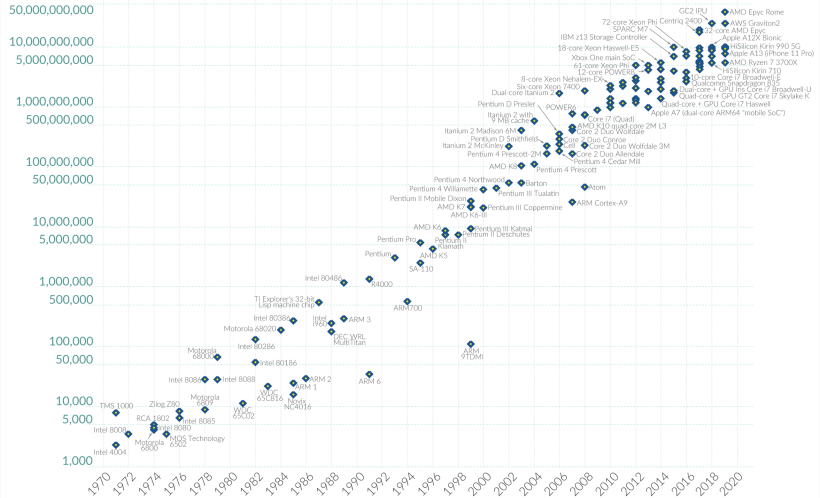
# History of Moore's Law

## Moore's Law: The number of transistors on microchips doubles every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Our World  
in Data

### Transistor count



Data source: Wikipedia ([wikipedia.org/wiki/Transistor\\_count](https://wikipedia.org/wiki/Transistor_count))

OurWorldinData.org – Research and data to make progress against the world's largest problems.

Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

## But wait...There's a Problem!

Transistors are starting to suffer power consumption and integrity issues:

- As the speed of transistors increases, their power consumption also increases.
  - Power consumption is dissipated as heat and when an integrated circuit gets hot it becomes unreliable.
- Transistor gates have become too thin, affecting their structural integrity, which leads to currents starting to leak.

## But wait...There's a Problem! (cont.)

- We are reaching the limits of how much air-cooled integrated circuits can effectively dissipate heat.
- Overall, its becoming impossible to continue to increase the speed of integrated circuits (hovering around 3.0GHz-3.7Ghz). Although with overclocking we've, seen this rise between 4.0GHz-4.9GHz.

Additionally physical manufacturing problems such as quantum tunneling (the inability to keep electrons contained beyond a certain thickness threshold) is also leading to a slow in single processor production.



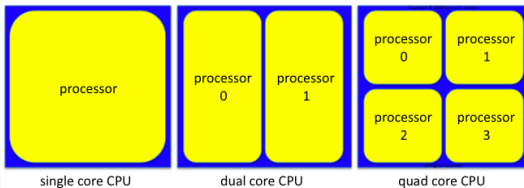
# Parallel Programming Overview

---

# Multi-core Processors to the Rescue

How can we continue to increase transistor density?: **Parallelism**

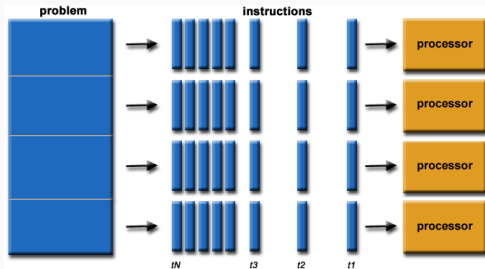
- No more complex and monolithic single processors.
- Instead the focus is on transitioning to multiple, simple and complete processors on one chip (multicore processors).
- Terminology:
  - Core : synonymous with central processing unit (CPU)
  - Multicore: more than one core on an integrated circuit.



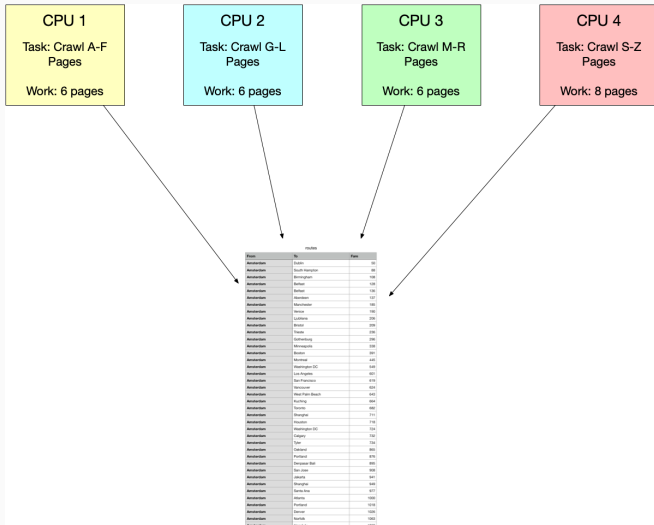
# What is Parallel Programming?

**Parallel Computing:** Simultaneously using multiple compute resources to solve a computational complex problem:<sup>2</sup>

- A problem is broken down into discrete parts to be executed concurrently.
- Each part contains their own set of instructions .
- Instructions from each part execute simultaneously on different processors.
- Synchronization between each part needs to happen to achieve determinism.

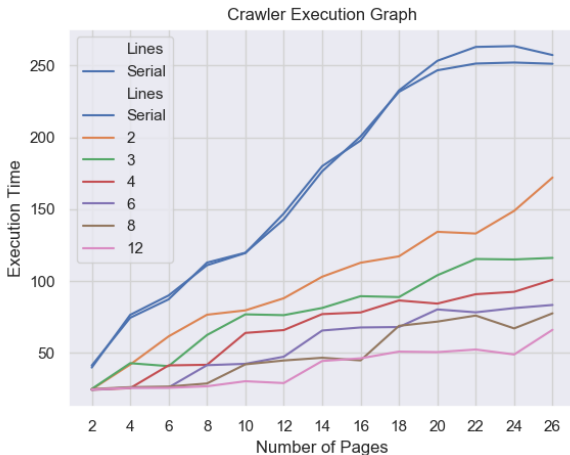


# Crawler Parallel Design



# Crawler Execution Time (Updated)

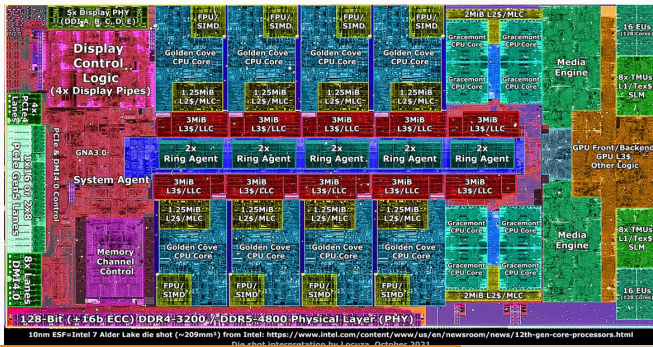
Execution time (in seconds) with a parallel component for our crawler:



# Real-World Multicore Architectures: Intel's Alder Lake-S Silicon

Die shot of the Alder Lake- processors. These are Intel's newest (January 2022) processors used in the Core i9-12900K, Core i7-12700K, and Core i5-12600K, and other Intel chips.

- 8-6 Performance CPU cores (normal CPU cores that are large, and run at high clock-speed)
- 8-4 Efficient CPU cores ( small CPU cores that run at reduced clock speed)



# Real-World Multicore Architectures: Apple Silicon



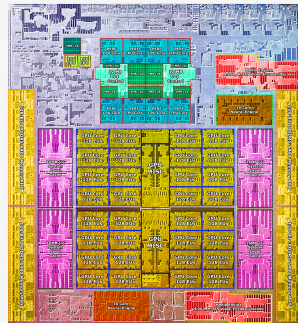
## Apple M1

- CPU:** 4x Performance Cores,  $\leq 3.2$  GHz  
+ 4x Efficiency Cores,  $\leq 2.06$  GHz
- GPU:** 8x GPU Cores (1024 EUs)  
24.576 „Threads in-flight“  
 $\sim 1.27$  GHz  
2.6 FP32 TeraFLOPs  
82 GigaPixel/s (64 TMUs)  
41 GigaPixel/s (32 ROPs)
- Memory:** 128-bit LPDDR4X-4266  
68.3 GB/s bandwidth



## Apple M1 Pro

- CPU:** 8x Performance Cores  
+ 2x Efficiency Cores
- GPU:** 16x GPU Cores (2048 Execution Units)  
49.152 „Threads in-flight“  
 $\sim 1.27$  GHz  
5.2 FP32 TeraFLOPs  
164 GigaPixel/s (128 TMUs)  
82 GigaPixel/s (64 ROPs)
- Memory:** 256-bit LPDDR5-6400  
204.8 GB/s bandwidth



## Apple M1 Max

- CPU:** 8x Performance Cores  
+ 2x Efficiency Cores
- GPU:** 32x GPU Cores (4096 Execution Units)  
98.304 „Threads in-flight“  
 $\sim 1.27$  GHz  
10.4 FP32 TeraFLOPs  
327 GigaPixel/s (256 TMUs)  
164 GigaPixel/s (128 ROPs)
- Memory:** 512-bit LPDDR5-6400  
409.6 GB/s bandwidth

## PS5

- 8x Zen2,  $\leq 3.5$  GHz
- 18x Cores (2304 EUs)  
92.160 „Threads“  
 $\sim 2.23$  GHz  
10.28 FP32 TeraFLOPs  
321 GigaPixel/s (144 TMUs)  
143 GigaPixel/s (64 ROPs)
- 256-bit GDDR6-14000  
448 GB/s bandwidth

Sources: M1 CPU clock frequency from Anandtech, chip images from Apple via Anandtech

Compiled and annotated by Locuza, October 2021

# Uses of Parallelism

More industries (industrial and commercial) in the world are requiring more computing power to solve complex tasks.

- **Weather forecast:** Based on complex mathematical models involving partial differential equations.
- **Crash Simulations:** car industry uses finite element methods to perform crash simulations.
- **Energy Research:** detailed models of technologies such as wind turbines, solar cells, and batteries can help construct more efficient clean energy sources.
- **Data Analysis** (“Big Data”, databases, Web search): We are generating tremendous amounts of data that need to be analyzed requires more computational power.
- **Computer Graphics:** more rendering power is needed to make more realistic effects in film.



# Parallel Programming Summary

In summary, what are the main reasons for parallel programming?

- Saving time and money
  - Having more resources reduces the time to complete a task, which saves money
  - Parallel computers can be built from cheap components
- Solve more complete problems
  - As I mentioned already, many problems are so large and/or complex that it is impractical or impossible to solve using serial computation.
- Provide concurrency:
  - A single compute resource can only do one thing at a time. Multiple compute resources can do many things simultaneously.

# Parallel Programming Summary (cont.)

- Take advantage of remote resources:
  - When local resources are scarce or insufficient then using remote resources (supercomputer, or interconnected computers over the internet).
- Make use of the parallel hardware:
  - Modern computers, even laptops, are parallel in architecture with multiple processors/cores.
  - Serial programs run on modern computers "waste" potential computing power.

In general, compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena.

# What will you learn in this course?

Main objectives:

- Understanding parallel architectures for fine tuning performance
- Synchronization mechanisms to maintain deterministic results
- Best practices for designing parallel programs
  - Partitioning at the task and data levels
- Work Distribution Techniques: algorithms, patterns and techniques to help with maintaining performance when scaling your application.

**Overall Goal:** Regardless of your programming language of choice (C, Java, Python etc.), you should be able to use the techniques, algorithms, patterns, and practices taught in this class and apply them to developing parallel applications in those languages.

# Tentative Roadmap

M9

Parallel Programming in on other platforms and languages  
(GPU Programming ,C, Python, etc.)

M8

Advanced Parallel Scheduling Techniques  
(Barriers, BSP models, Strong Scaling vs Weak Scaling)

M7

Advanced Parallel Scheduling Techniques  
(Future, Scheduling, Work-Distribution & Stealing )

M6

Concurrent Execution Models & Design  
(Concurrency in Go, Thread execution models)

M5

Designing Parallel Programs & Performance  
(Best practices, data dependencies, performance analysis )

M4

High-level Synchronization Primitives  
(Concurrent Objects: Lists, Queues, Stacks, Hash tables)

M3

Theoretical Principles & Low-Level Synchronization Primitives  
(Critical Sections, Locks, Monitors, Semaphores, Barriers)

M2

Hardware Basics/Architecture  
(Cache, Interconnect, Processes/Threads, Atomic Operations)

## Aside: Classical Use of Parallelism

**History:** Parallel programming and design of efficient parallel programs is well established in high performance computing (HPC) for many years.

- Specifically for solving scientific problems using simulations.
- More precise simulation(s) of larger problems need greater computing power and space.
- HPC research and developed led to new developments in parallel hardware and software technologies.
- Hardware: cluster systems that are built up from server nodes where computations to be preformed are partitioned into parts and are assigned to parallel resources.



## Video: Go Bootcamp (Homework 1)

---

# Design of Go

Go was designed by Google (specifically Rob Pike, Kenneth Thompson in 2007) to solve problems that Google faces.

## Goals

- Eliminate slowness
- Inefficiencies
- Maintain and improve scale

## Types of problems Google faces?

- C++ for servers, plus lots of Java and Python mixed in
- Large employee base
- Millions upon Millions of lines of code for various projects
- Distributed build system
- Millions of compute cluster machines, needing to perform tasks

# Who uses Go?

## Trends - Major Companies using Golang



3

<sup>3</sup><https://www.quora.com/>

*Other-than-Google-what-companies-are-using-Go-in-production*



# How are we going to learn Go?

We are actually going to learn Go by going through examples shown on: *<https://gobyexample.com>*

**Syntax** you should know:

- Values
- Variables
- Constants
- Importing
- For
- If/Else
- Switch
- Arrays and Slices
- Maps
- Range
- Functions
- Multiple Return Values
- Variadic Functions
- Pointers
- Structs
- String Formatting and Functions