Welcome and Introduction

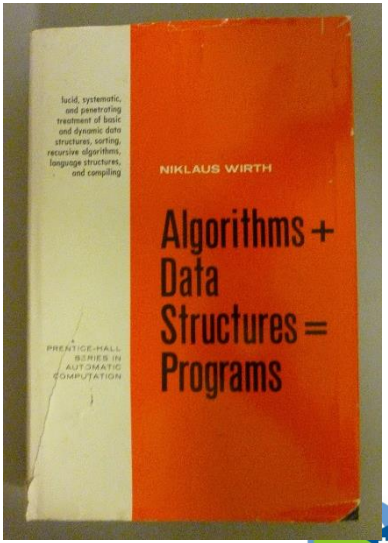
Lecture 1

Hartmut Kaiser

https://teaching.hkaiser.org/spring2025/csc4700/

Course Overview

- Programs == Algorithms + Data structures
- Selecting the right data structure
  - Selecting containers: trivial
  - Selecting data types: type properties
- Selecting and developing algorithms
  - Algorithmic thinking
  - Parallelization almost an afterthought



Administrativia

Admin & Organizational

- Course:
  - Depth first introduction, C++ Standard Library, C++ Data structures and algorithms, parallel algorithms, scientific applications, architectures, parallel programming models…
  - https://teaching.hkaiser.org
  - hkaiser@cct.lsu.edu
  - Discord server: https://discord.gg/Q32sGfKqMS
- Reading:
  - Diehl's Parallel C++
  - Stepanov's From Mathematics to Generic Programming
  - Koenig's Accelerated C++
  - Stroustrup's Programming - Principles and Practice Using C++
- Homework, project, quizzes, grading, honesty
  - Use Github classroom for assignments
  - Use VSCode as a tool

Important Dates

- Lectures
  - Tuesday and Thursday, 10:30 to 11:50, 1212 PFT
- Grading
  - Homework 50%
  - Project 25%
  - Midterm exam 10%
  - Final exam 15%
- Exams
  - Midterm exam: March 6
  - Final exam: May 8, 5:30pm - 7:30pm

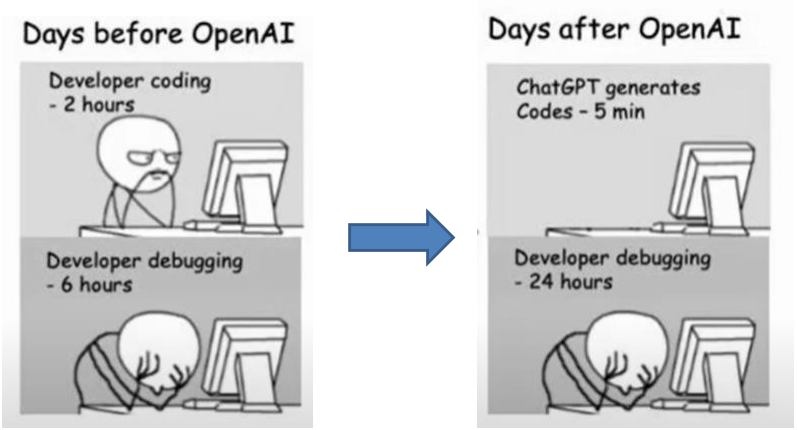
Ground Rules

- All information about the course: teaching.hkaiser.org
  - Following that site is expected and will positively impact overall grade
- Lectures
  - Attendance is expected and will positively impact overall grade
- Assignments
  - Up to five individual assignments focusing on writing code
    - Use of (standard C++) algorithms and (standard C++) data structures
    - Use of various parallel programming models
  - Assignment 0 has been posted (due: January 27, 11:59 pm)
- All assignments and the project are hosted on GitHub (github.com) and managed through GitHub classroom (classroom.github.com)
  - You will need to create an account on Github

Honesty

- The LSU *Code of Student Conduct* defines plagiarism in Section 5.1.16:
  - "Plagiarism is defined as the unacknowledged inclusion of someone else's words, structure, ideas, or data. When a student submits work as his/her own that includes the words, structure, ideas, or data of others, the source of this information must be acknowledged through complete, accurate, and specific references, and, if verbatim statements are included, through quotation marks as well. Failure to identify any source (including interviews, surveys, etc.), published in any medium (including on the internet) or unpublished, from which words, structure, ideas, or data have been taken, constitutes plagiarism;"
- Plagiarism will not be tolerated and will be dealt with in accordance with and as outlined by the LSU Code of Student Conduct: https://www.lsu.edu/saa/students/codeofconduct.php

ChatGPT?


Days before OpenAI
Developer coding
- 2 hours
Developer debugging
- 6 hours
Days after OpenAI
ChatGPT generates
Codes - 5 min
Developer debugging
- 24 hours

ChatGPT

- The goal for this course is to train our brains, not ChatGPT
  - So, do yourself a favor and don't use it
  - There are too many software 'developers' out there who copy & paste their way to the next paycheck
- However, if you do use it:
  - Never use anything without carefully reviewing it
    - Assume what you got is wrong! Prove to yourself it is correct!
  - The skill of reading (and understanding) code becomes more important than ever
  - Cite and annotate the copied code
- Whatever you do, remember:
  - It's plagiarism if you submit the same code as your neighbor
  - No matter where you got it from, be it Google, ChatGPT, or your neighbors computer

How to Learn

- Ask yourself at the beginning of this semester
  - What scientific problem you want to solve?
  - What do you want to learn from this course that can prepare you?
  - Can you write a sequential program to solve it?
  - What is the performance of it?
  - …

- Ask yourself at the end of this semester
  - Did you master the skillset to solve your scientific problem?
  - Can you write a high-performance program to solve it?
  - What is the performance of it?
  - What is the speedup?
    - Compare your sequential program with your high-performance program

Tour of the Course

- Basic CPU machine model
  - Hierarchical memory (registers, cache, virtual memory)
  - Instruction level parallelism
  - Multicore processors
- Scientific computing
  - Linearization of physics model
- Shared memory parallelism
- GPU
- Distributed memory parallelism

- Use running examples to explain concepts and APIs

Tour of the Course

- Elements of C++
- Elements of software organization
- Elements of software practice
- Elements of performance measurement and optimization

What This Course is About

- How algorithms, data, software, and hardware interact to affect performance (and how to orchestrate them to get high performance)

- At the completion of this course, you will be able to
  - Write software that fully utilizes hardware performance features
  - Describe the principal architecture mechanisms for high performance and algorithmic and software techniques to take advantage of them
  - Recognize opportunities for performance improvement in extant code
  - Describe a strategy for tuning HPC code

- Today and years from now

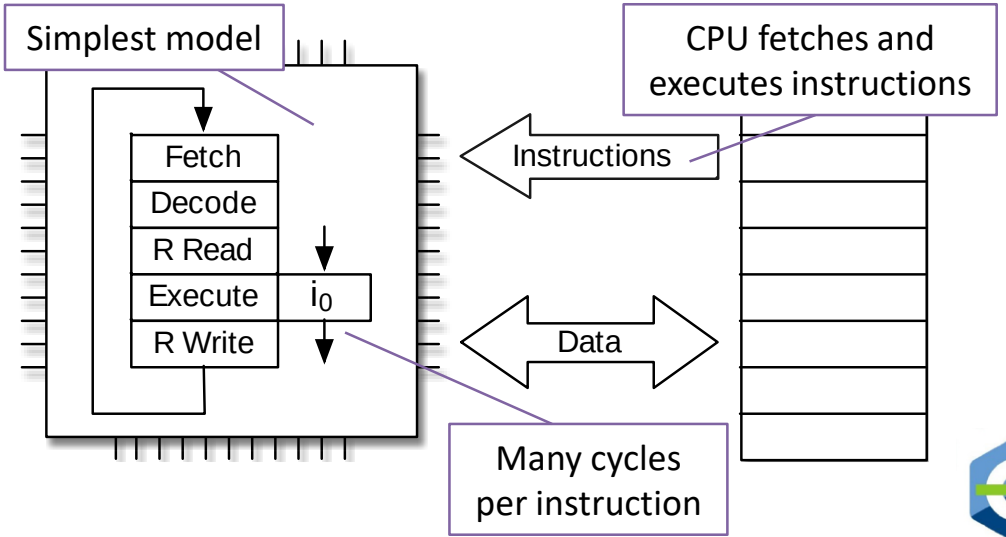
Computing is Indispensable to Science and Engineering

- The 3rd (and 4th?) pillar(s)
- Can carry out investigations where physical experiments would be too fast, too slow, too hot, too cold, too costly, too dangerous, etc.
- Examples: Weather, climate, fusion, crash testing, etc.
- HPC means more and better scientific discovery
- Better world, survival of the planet

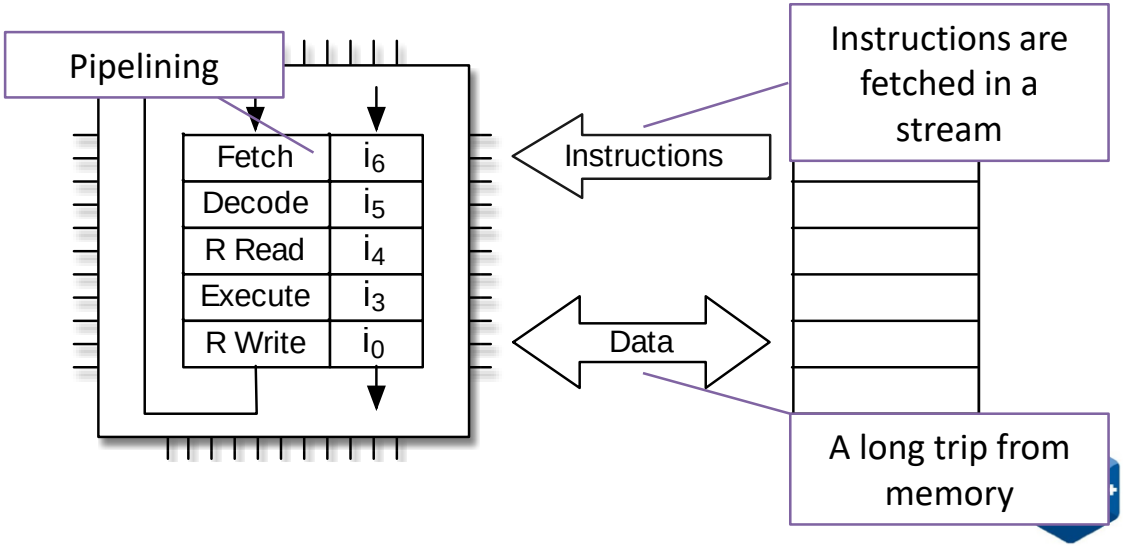
The HPC Canon (as of 2025)

Technology	Paradigm	Hammer
CPU (single core)	Sequential	C compiler
SIMD/Vector (single core)	Data parallel	Intrinsics (asm)
Multicore	Threads	pthreads (C) library
NUMA shared memory	Threads	pthreads (C) library
GPU	GPU	CUDA
Clusters	Message passing	MPI (C) library

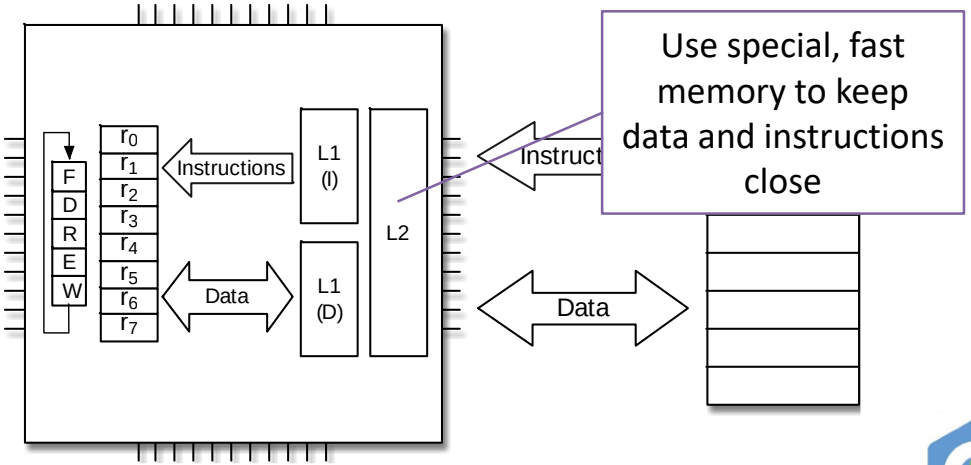
Scaling progression of CPUs



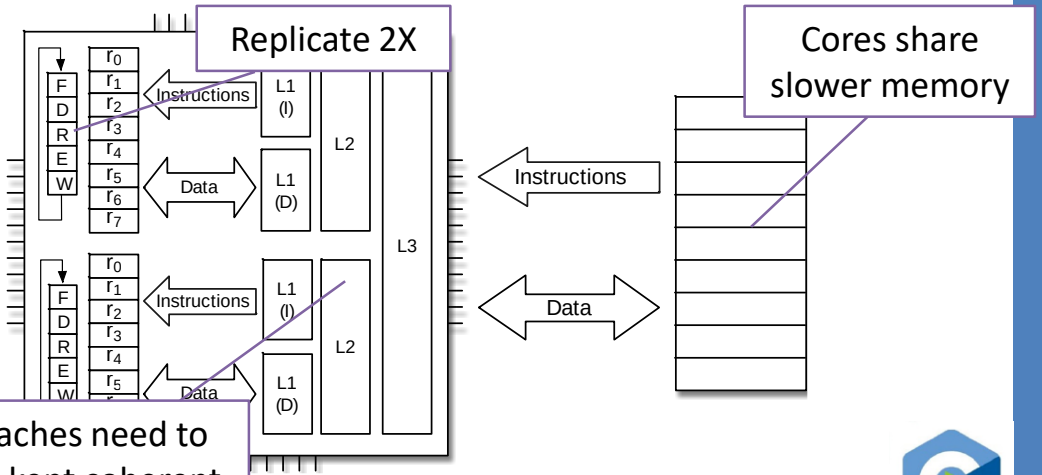
Pipelining


Pipelining
Fetch
Decode
R Read
Execute
R Write
i_6
i_5
i_4
i_3
i_0
Instructions
Data
Instructions are fetched in a stream
A long trip from memory

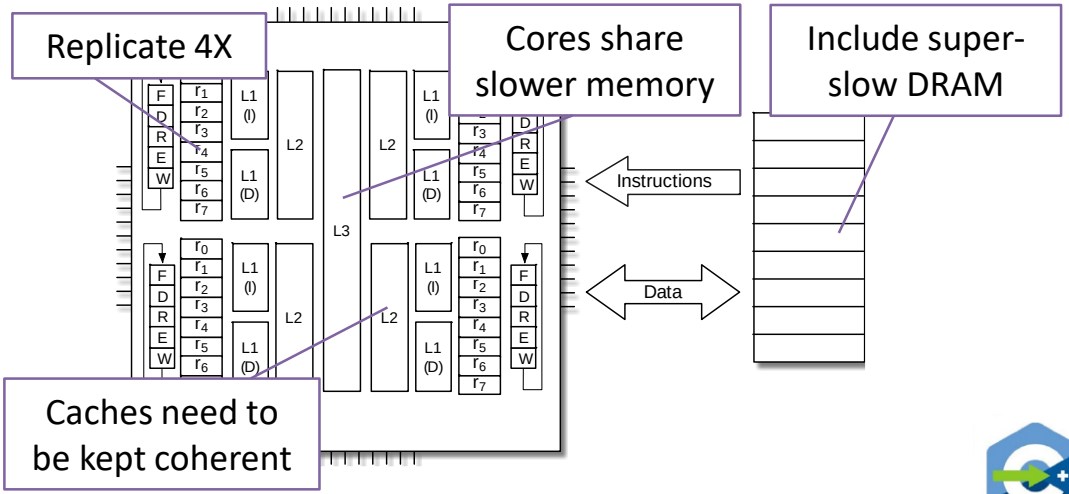
Hierarchical Memory



Multicore CPUs



Even more Cores

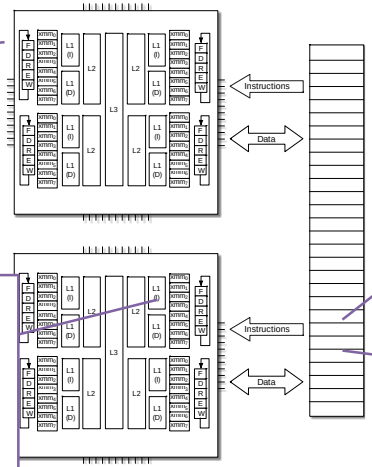


Symmetric Multi-Processor (SMP)

Multiple CPU chips

AKA “sockets”

Caches still need to be kept (somewhat) coherent



Memory may be uniformly shared among sockets

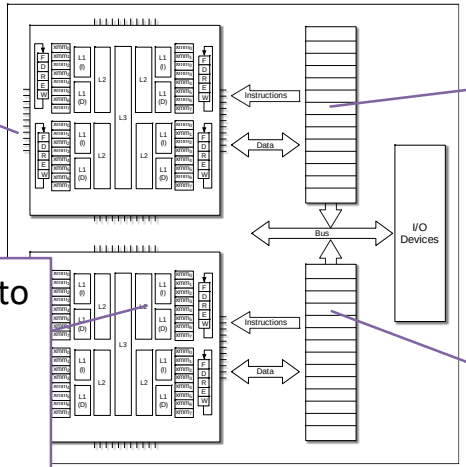
Uniform memory access (UMA)

Asymmetric Multi-Processor

Multiple CPU chips

AKA “sockets”

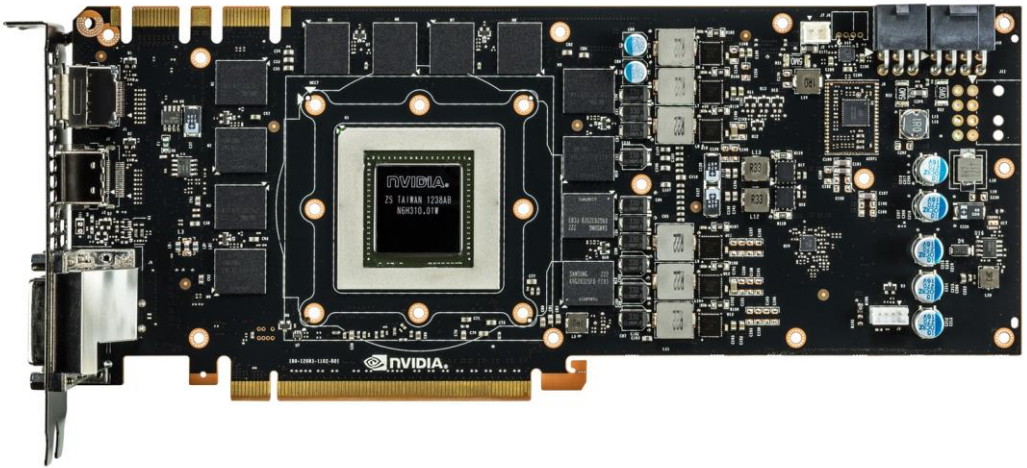
Caches still need to be kept (somewhat) coherent: CC-NUMA



Memory may be non- uniformly shared among sockets

Non-uniform memory access (NUMA – most common)

GPU



The Next Step

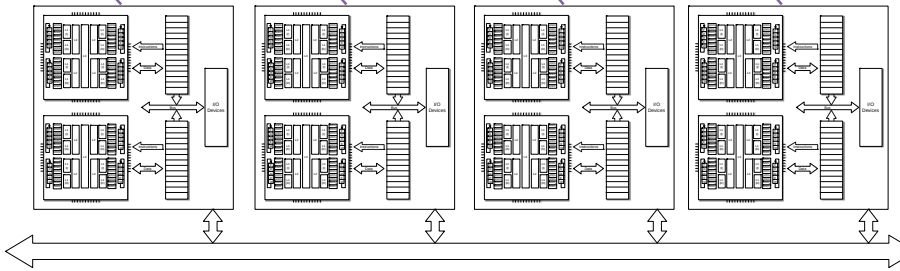
Put sockets on a blade

Put blades in a chassis

Put chassis in a rack

Put racks in a center

Put centers in the cloud



Then you have a Supercomputer

But how do you use it?

The HPC Canon (as of 2025)

Technology	Paradigm	Hammer
CPU (single core)	Sequential	
SIMD/Vector (single co	Data parallel	
Multicore	Threads	
NUMA shared memory	Threads	
GPU	GPU	
Clusters	Message passing	

This semester

C++

Editorial Comment

- The most exciting phrase to hear in science, the one that heralds new discoveries, is not “Eureka!” (I found it) but “That’s funny”
  - Attributed to Isaac Asimov (and others)

Uses of HPC (a Sample)

- Cosmology
- Earthquake
- Weather
- Climate modeling
- Automobile crash testing
- Aircraft design
- Jet engine design
- Stockpile stewardship
- Nuclear fusion
- Protein folding
- Modeling the brain
- Modeling bloodstream
- Epidemiology
- Rendering (CGI)
- Sigint
- Block chains
- Gene sequencing
- Etc

Multiphysics Solver

$$\psi = \psi_e(\boldsymbol{F}_e, T) + \psi_p(\boldsymbol{\chi}, T) + \psi_T(T)$$

$$-\rho_0 T\left(\frac{\partial \dot{\psi}_T}{\partial T}\right) + \mathrm{Div}\boldsymbol{Q} = Q_p + Q_e + Q_c + \rho_0 r$$

$$Q_p = \boldsymbol{F}_e^T \boldsymbol{P}\left[\boldsymbol{F}_T^T - T\left(\frac{\partial \boldsymbol{F}_T}{\partial T}\right)^T\right] : \dot{\boldsymbol{F}}_p - \frac{\partial \psi_p}{\partial \boldsymbol{\chi}} \cdot \dot{\boldsymbol{\chi}} -$$

$$- T\left[\boldsymbol{P}\left(\boldsymbol{F}_p^{-T} : \dot{\boldsymbol{F}}_p\right) - J_p J_T \frac{\partial W_e}{\partial \boldsymbol{F}_e} \boldsymbol{F}_p^{-T} \dot{\boldsymbol{F}}_p^{\,T} \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T}\right] : \boldsymbol{F}_e \boldsymbol{F}_p \frac{\partial \boldsymbol{F}_T}{\partial T} + T\rho_0 \left(\frac{\partial \dot{\psi}_p}{\partial T}\right)$$

$$Q_e = -T\left[\boldsymbol{P}\left(\frac{\partial \boldsymbol{F}_T}{\partial T}\right)^T \boldsymbol{F}_p^{-T} : \dot{\boldsymbol{F}}_e + \left(J_p J_T \frac{\partial^2 W_e}{\partial \boldsymbol{F}_e \partial \boldsymbol{F}_e} : \dot{\boldsymbol{F}}_e \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T}\right) : \boldsymbol{F}_e \boldsymbol{F}_p \frac{\partial \boldsymbol{F}_T}{\partial T}\right] + T\rho_0 \left(\frac{\partial \dot{\psi}_e}{\partial T}\right)$$

$$Q_c = -T\left\{\left[\boldsymbol{P}\left(\boldsymbol{F}_T^{-T} : \frac{\partial \boldsymbol{F}_T}{\partial T}\right) - \boldsymbol{P}\left(\frac{\partial \boldsymbol{F}_T}{\partial T}\right)^T \boldsymbol{F}_T^{-T} + J_p J_T \frac{\partial^2 W_e}{\partial \boldsymbol{F}_e \partial T} \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T}\right] : \boldsymbol{F}_e \boldsymbol{F}_p \frac{\partial \boldsymbol{F}_T}{\partial T} + \right.$$

$$\left. + \boldsymbol{P} : \boldsymbol{F}_e \boldsymbol{F}_p \frac{\partial^2 \boldsymbol{F}_T}{\partial T \partial T}\right\} \dot{T}$$

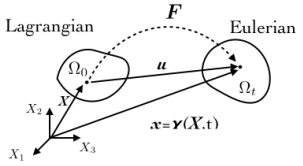
$$\boldsymbol{P}\Big|_{\Omega_0} = \rho_0 \frac{\partial \psi_e}{\partial \boldsymbol{F}_e} \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T} = J_p J_T \frac{\partial W_e}{\partial \boldsymbol{F}_e} \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T}$$

$$\eta = -\frac{\partial \psi}{\partial T}\Big|_{\boldsymbol{F}} = \frac{1}{\rho_0} \boldsymbol{P} : \boldsymbol{F}_e \boldsymbol{F}_p \frac{\partial \boldsymbol{F}_T}{\partial T} - \frac{\partial \psi_e}{\partial T} - \frac{\partial \psi_p}{\partial T} - \frac{\partial \psi_T}{\partial T}$$

$$\boldsymbol{Q} = -\boldsymbol{F}^{-1} \boldsymbol{\kappa} \boldsymbol{F}^{-1} \nabla_0 T$$

- Multiplicative split

$$\boldsymbol{F} = \boldsymbol{F}_e \boldsymbol{F}_p \boldsymbol{F}_T$$



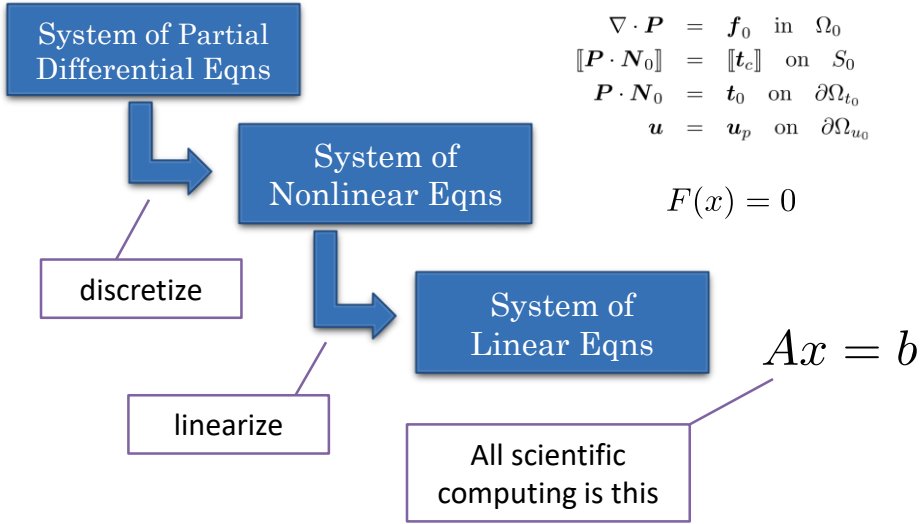
x

Physics: Systems of Partial Differential Equations (PDEs)

Elliptic	Parabolic	Hyperbolic
$\nabla \cdot \boldsymbol{P} = \boldsymbol{f}_0$ in Ω_0 $[\![\boldsymbol{P} \cdot \boldsymbol{N}_0]\!] = [\![\boldsymbol{t}_c]\!]$ on S_0 $\boldsymbol{P} \cdot \boldsymbol{N}_0 = \boldsymbol{t}_0$ on $\partial\Omega_{t_0}$ $\boldsymbol{u} = \boldsymbol{u}_p$ on $\partial\Omega_{u_0}$	$\rho_0 c\dot{T} - \nabla_X \boldsymbol{Q} = Q_f$ on Ω_0 $[\![\boldsymbol{Q} \cdot \boldsymbol{N}_0]\!] = 0$ on S_0 $\boldsymbol{Q} = \boldsymbol{Q}_p$ on $\partial\Omega_{Q_0}$ $T = T_p$ on $\partial\Omega_{T_0}$	$\rho_0 \frac{\partial^2 \boldsymbol{u}}{\partial \boldsymbol{u}^2} - \nabla \cdot \boldsymbol{P} = \boldsymbol{f}_0$ in Ω_0 $[\![\boldsymbol{P} \cdot \boldsymbol{N}_0]\!] = [\![\boldsymbol{t}_c]\!]$ on S_0 $\boldsymbol{P} \cdot \boldsymbol{N}_0 = \boldsymbol{t}_0$ on $\partial\Omega_{t_0}$ $\boldsymbol{u} = \boldsymbol{u}_p$ on $\partial\Omega_{u_0}$ $\boldsymbol{u}(0) = \boldsymbol{u}_0$ in Ω_0 $\dot{\boldsymbol{u}}(0) = \boldsymbol{v}_0$ in Ω_0
• constitutive law hyperelastic multiscale	• constitutive law Fourier's law	• constitutive law hyperelastic multiscale
• state variables damage visco-elastic	• state variables porosity chemical reactions	• state variables damage visco-elastic
$\rho_0 = J\rho$	•mass conservation based on physics	$\rho_0 = J\rho$
• solution strategy sparse iterative solver dual domain dec.	•solution stratedy sparse iter. solver α-method integrator	• solution strategy sparse iterative solver dual domain dec. MD-AVI

Courtesy Karel Matous, U. Notre Dame

Computational Science



C++! But Why?

Why C++ ?

- You can't learn to program without a programming language
- The purpose of a programming language is to allow you to express your ideas in code
- C++ is the language that most directly allows you to express ideas from the largest number of application areas
- C++ is the most widely used language in engineering areas
  - http://www.research.att.com/~bs/applications.html
- Our society runs on software
  - Large parts of that software directly or indirectly rely on C/C++
    - Windows, Linux, Office Suite, Adobe Products, …
    - Self driving cars, industry equipment, …
    - Internet

Why C++ ?

- C++ is precisely and comprehensively defined by an ISO standard
  - C++17! Now also C++23!
  - And that standard is almost universally accepted
- C++ is available on almost all kinds of computers
- Programming concepts that you learn using C++ can be used fairly directly in other languages
  - Including C, Java, C#, and (less directly) Fortran
- Last but not least: C++ jobs are one of the best paid jobs available

A First C++ Program

```
// Our first C++ program

#include <algorithm>
#include <print>
#include <vector>

int main()     // main() is where a C++ program starts
{
    std::vector<double> data = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 };
    auto sum = std::reduce(data.begin(), data.end());
    std::print(std::stdout, "Sum of all values: {}\n", sum);
    return 0;
}
```

A deeper Look

- Expressions
  - Compute something, yields result, may have side effects
  - Operands and operators – types!
- Scope
  - Part of the program in which a name has its meaning
    - Global scope
    - Namespace scope
    - Block scope
- Algorithms

```
auto sum = std::reduce(data.begin(), data.end());
std::println(std::stdout, "Sum of all values: {}", sum);
```

Details

- Program structure
  - Free form except string literals, #include, comments
- Types
  - Data structures and their operations
  - Built in and user defined
- Functions
  - Code structure
  - Helps isolating (abstracting) sub-functionalities
- Namespaces
  - Grouping related names

Details

- Special character literals
  - `\n` newline character
  - `\t` horizontal tabulator
  - `\b` backspace character
  - `\"` same as " but does not terminate string
  - `\'` same as ' but does not terminate character literal
  - `\\` same as \ but does not give special meaning to next character
- Definitions and headers
- The main() function
- Braces and semicolons
- Output

Why Parallel C++?

- Modern computers have more than one core
  - Often many more
  - Many programs require to do many things at once (MTAO)
- Scientific computing requires a lot of number crunching
  - Parallelization required to reduce required execution times
- C++ offers
  - Parallel algorithms
    - Divide a loop into subtasks, which are processed simultaneously
  - Task and Data parallelism
    - Focuses on distributing tasks across cores
    - Divides data arrays into smaller chunks and processes them in parallel
  - Asynchronous execution

Motivation for Parallelism

- Operating systems must handle multiple things at once (MTAO)
  - Processes, interrupts, background system maintenance
- Networked servers must handle MTAO
  - Multiple connections handled simultaneously
- Parallel programs must handle MTAO
  - To achieve better performance
- Programs with user interface often must handle MTAO
  - To achieve user responsiveness while doing computation
- Network and disk bound programs must handle MTAO
  - To hide network/disk latency
  - Sequence steps in access or communicatoin

A First Parallel C++ Program

```
// Our first parallel C++ program

#include <algorithm>
#include <execution>
#include <print>
#include <vector>

int main()     // main() is where a C++ program starts
{
    std::vector<double> data = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 };
    auto sum = std::reduce(std::execution::par, data.begin(), data.end());
    std::println("Sum of all values: {}", sum);
    return 0;
}
```

A deeper Look

- Parallel algorithms
  - `std::execution::par`: execution policy tells library that it is ok to execute out of order

```
auto sum = reduce(std::execution::par, data.begin(), data.end());
```

- This is a hint to the library
  - Implementation is free to decide what to do
  - No real control over how things are being executed

A First Parallel HPX Program

```
// Our first parallel HPX program

#include <hpx/algorithm.hpp>
#include <hpx/execution.hpp>
#include <print>
#include <vector>

int main()     // main() is where a C++ program starts
{
    std::vector<double> data = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 };
    hpx::execution::experimental::num_cores nc(2);
    auto sum = hpx::reduce(hpx::execution::par.with(nc), data.begin(), data.end());
    std::println("Sum of all values: {}", sum);
    return 0;
}
```

A deeper Look

- Parallel algorithms
  - `hpx::execution::par`: execution policy tells library that it is ok to execute out of order
  - `hpx::execution::experimental::num_cores` encapsulates a way to control number of cores to be used
  - `.with(nc)`: associates this with the execution policy

```
hpx::execution::experimental::num_cores nc(2);
auto sum = hpx::reduce(
    hpx::execution::par.with(nc), data.begin(), data.end());
```

- This makes sure that the algorithm runs on 2 cores only

Hello, world!

- "Hello world" is a very important program
  - Its purpose is to help you get used to your tools
    - Compiler
    - Program development environment
    - Program execution environment
  - Type in the program carefully
    - After you get it to work, please make a few mistakes to see how the tools respond; for example
      - Forget the header
      - Forget to terminate the string
      - Misspell return (e.g. retrun)
      - Forget a semicolon
      - Forget { or }
      - …

Number of Unique Elements

- Let's start with a sequence of integers:

```
int a[] = { 1, 3, 1, 4, 1, 5 };
```

- How many unique integers do we have?
  - Simple solution:

```
#include <iostream>
#include <set>

int main() {
    std::set<int> set_of_ints(a, a + 6);
    std::println("{}", set_of_ints.size());
}
```

- This solution is correct! But … very slow! Why?

Equality vs. Ordering

- `std::set` is implemented using Red-Black-Trees, which according to textbooks is the best way of implementing it
  - Will do $O(n \log(n))$ comparison operations, but with a large constant coefficient
  - Has to re-sort whole data structure for each insertion without need
- It might appear, that finding unique elements does not require ordering, it just requires equality
  - But, actually we need a search or find
  - Equality gives us linear search $O(n)$, while sorting gives us binary search $O(\log(n))$ so we can find much, much faster.
- Correct solution is to use `std::unique`

Number of Unique Elements

- Sequence must be sorted, however. `std::unique` eliminates equal elements and returns reference to first duplicated element:

```
#include <algorithm>
#include <iostream>

int main() {
    std::sort(a, a + std::size(a));
    std::println("{}", std::unique(a, a + std::size(a)) - a);
}
```

- For example:

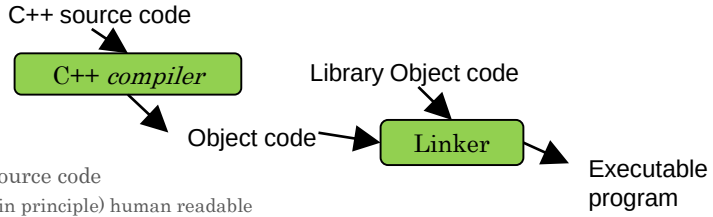
```
1 2 2 2
    ^
```

Use the correct STL Data Structures

- What container type should you use?
  - Whenever you can, use `std::vector`.
  - If you cannot, find a way so you can.
- Avoid any data structures except arrays. “Well aren’t there exceptions?” No, not for you.
  - Typically advanced data structures are slower than simple data structures.
  - Data structures which appear to be alright when textbook writers wrote their books, are no longer all right now.
  - Computers changed so much. Things no longer work as taught 10 years ago.
- Occasionally we will ask you to use other data structures
  - Mostly to make our point, though

A Word about Compilation

Compilation and Linking



- You write C++ source code
  - Source code is (in principle) human readable
- The compiler translates what you wrote into object code (sometimes called machine code)
  - Object code is simple enough for a computer to "understand"
- The linker links your code to system code needed to execute
  - E.g. input/output libraries, operating system code, and windowing code
- The result is an executable program
  - E.g. a .exe file on windows or an a.out file on Unix

See: Decoding C++ Compilation Process: From Source Code to Binary

CMake

- CMake is a family of tools
  - Building software
  - Testing software
  - Packaging software
- We will use it for building and testing
- CMake generates build systems files (Makefiles and or workspaces)
  - Those can be used to automatically build and test your code
- The user writes a single set of descriptive scripts
  - Define Targets and their inter-dependencies
- CMake is well integrated with many IDEs

Simplest Example

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.0)
project(Demo1)
add_executable(Demo1 demo1.cpp)
```

In source build:

```
% cd demo1
% ls
CMakeLists.txt  demo1.cpp
```

Simplest Example

```
% cmake .
-- The C compiler identification is GNU 7.3.0
-- The CXX compiler identification is GNU 7.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile fe
-- Detecting C compile fe
-- Check for working CXX
-- Check for working CXX
-- Detecting CXX compiler
-- Detecting CXX compiler
-- Detecting CXX compile
-- Detecting CXX compile
-- Configuring done
-- Generating done
-- Build files have been
```

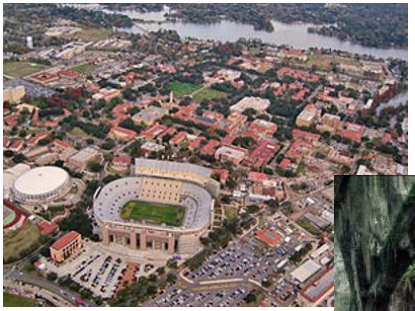
```
% make
Scanning dependencies of target Demo1
[ 50%] Building CXX object
CMakeFiles/Demo1.dir/demo1.cpp.o
[100%] Linking CXX executable Demo1
[100%] Built target Demo1

% ./Demo1
Hello, world!
```

Homework and Projects

- We will use Visual Studio Code as our development environment
  - Assignment 0 will ask you to install everything and make it work
  - Alternatively use Github Codespaces (development fully online)
- We will use Github Classroom for managing your submission
  - Please go get a login on github.com, if you have none yet (this is part of assignment 0)
  - This will teach you the use of the Git source code control system along the lines
  - Special lecture will talk about development environment (git, cmake, etc.)

Summary

- Today, single-thread-of-control performance is improving very slowly
  - To run programs significantly faster, programs must utilize multiple processing elements or specialized processing hardware
  - Which means you need to know how to reason about and write parallel and efficient code
- Writing parallel programs can be challenging
  - Requires problem partitioning, communication, synchronization
  - Knowledge of machine characteristics is important
  - In particular, understanding data movement!
- I suspect you will find that modern computers have tremendously more processing power than you might realize, if you just use it efficiently!



LSU CENTER FOR COMPUTATION & TECHNOLOGY

STE||AR GROUP