Introduction

Computational Science II (CAAM 520)

Christopher Thiele

Rice University, Spring 2020

General information

Course:	Computational Science II (CAAM 520)
Class:	MWF 1:00pm – 1:50pm in DCH 1075
Instructor:	Christopher Thiele
Office:	DCH 3015
Email:	ct37@rice.edu
Office hours:	W 2:00pm – 3:00pm in DCH 3110

→ All information on class website:
https://cthl.github.io/caam520

Motivation

Today's computers have multi-core CPUs that cannot be utilized by ***sequential***, i.e., non-parallel software.

- This is true even for laptop computers, phones, etc.
- Tools and paradigms are needed to use such hardware.
- We will learn about ***shared memory parallelism***
  $\longrightarrow$ OpenMP

Motivation

Multiple computers will always be faster than a single computer.

- Large scientific applications require computer ***clusters***.
- Requires ***additional*** tools and paradigms.
- We will learn about ***distributed memory parallelism*** $\longrightarrow$ MPI (Message Passing Interface)

Motivation

Even though parallelism makes programming more difficult, we must develop software that is

- correct,
- robust,
- maintainable,
- extensible, and
- portable.

→ We will touch on aspects of software engineering where appropriate.

Overview

In this course, we will learn to

- understand the benefits and challenges of parallel computing on current multi-core computers,
- design parallel algorithms for scientific applications,
- implement parallel algorithms with OpenMP, MPI, and hybrid approaches,
- assess the suitability of parallel programming techniques for scientific applications,
- measure the performance and scalability of scientific software, and
- develop robust, extensible, and maintainable scientific code.

Tentative list of topics

- Recap of C (specifically pointers, memory management)
- Standard development tools (GCC, make, GDB, gprof, Valgrind, Git)
- CPU and memory architecture on current multi-core systems
- Parallelism – some theory
- Introduction to shared memory parallelism with OpenMP
- Introduction to distributed memory parallelism with MPI
- Hybrid parallelism

Tentative list of topics (optional)

- Advanced MPI (non-blocking and one-sided communication)
- Scientific software at scale (PETSc)

Texts and materials

There are no required texts for this course.

The following texts and manuals are recommended:

- Introduction to High Performance Computing for Scientists and Engineers by Hager and Wellein (2011)
- The C Programming Language (2nd ed.) by Kernighan and Ritchie (1988)

→ See website for additional material.

Prerequisites

- You should be familiar with C and the Linux/Unix command line.
- You should have basic knowledge of Git.
- You must have access to a computer on which you can install software, preferably a laptop computer.
- You must have (or create) a GitHub account.

Grading

- Grades will be based on homework submissions.
- Some homework problems might be pledged.
- There will not be any exams.

Homework

Homework will involve programming and writing a report.

- Anticipate a homework every 2 – 3 weeks (one per section).
- Submit code and reports to a private GitHub repository (will be provided).
- Late submissions will incur a penalty of 10% per day. Ask ***in advance*** for exceptions.

Homework

Reports must be submitted as PDF files.

Code can be written in C or C++ (any standard) as long as

- a Makefile is provided, and
- interfaces are not violated.

Homework

Example 1: The task is to implement a function

```
int norm2(const double *x, int n)
```

You can implement this function in C or C++, but you must not change its name or signature, i.e., do not implement

```
double euclidean_norm(const std::vector<double> &x)
```

Homework

Example 2: The task is to implement a structure

```
typedef struct
{
  double *x;
  int n;
} my_vector;
```

You can implement this function in C or C++, but do not implement

```
class my_vector
{
private:
  double *x;
  int n;
};
```

Homework

A Linux-based virtual machine (VM) with all software required for the homework is available here.

It can be used with VirtualBox (https://www.virtualbox.org/) on Linux, macOS, and Windows.

Use of the VM is strongly encouraged.

- Some software is difficult to install on some systems, e.g., GCC and GDB on macOS.
- Using the VM ensures that your code works on the grader's system.