Scientific Computing for Physicists

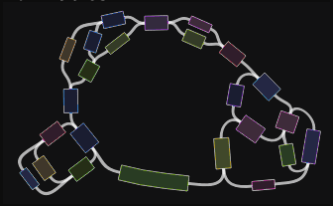
Ramses van Zon

PHY1610H 2025 Winter

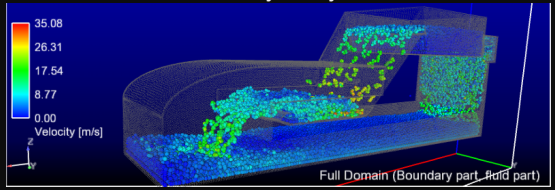
Course Intro

Examples of Scientific Computations

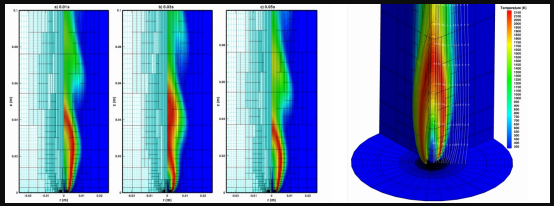
- BioInformatics



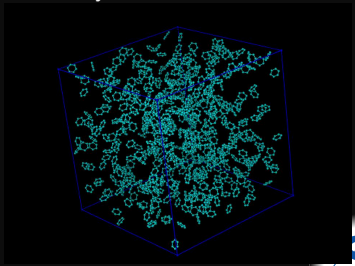
- Smooth Particle Hydrodynamics



- Computational Fluid Dynamics



- Molecular Dynamics



Course Topics

This course aims at making you a more productive and efficient computational scientist.

It will cover best practices in scientific computing and programming skills, optimization and a bit of parallel programming.

There are three main themes in this course:

1. Scientific Software Development
2. Numerical Tools for Physical Scientists
3. High Performance Scientific Computing

Your Instructor

- My name is Ramses van Zon
- I am a High-Performance Computing Analyst at the SciNet HPC Consortium here at the University of Toronto.
- After a Ph.D. in Mathematical Physics, I postdoc-ed in Chemical and Theoretical Physics, which included development of molecular dynamics simulations and other computational projects.
- Nowadays, I'm involved in training and education and various aspects of running and supporting "high performance computing".
- The TA for this course is Kayhan Momeni. He'll be helping with the grading of the assignments. He has taken this course in the past, so he knows what you're going through.

What is SciNet?

SciNet is UofT's supercomputer centre, which hosts and supports one of Canada's fastest supercomputers available to academic researchers.

https://www.scinethpc.ca

We also do a lot of other teaching (Bash, Python, R, Fortran, C++, GPU programming, databases, machine learning, parallel programming, ...)

https://scinet.courses

On a national level, we are a partner of the Digital Research Alliance of Canada (the successor of the Compute Canada Federation).

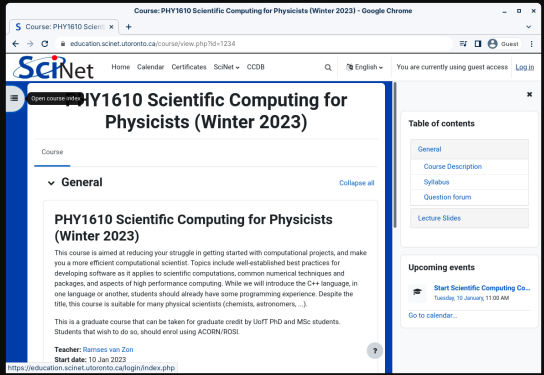
Course website

https://scinet.courses/1396

- Lectures
- Recordings
- Assignments
- Forum

Near-weekly assignments posted on the site on Thursdays.

To be able to hand in assignments to the course website, you need to be able to login to the site (use your Alliance/CCDB account if you have one).



Accounts for this course

- If you do not have an Alliance account, your login name on the course site is something that starts with `tmp_...`
- For assignments, you'll have access to SciNet's Teach cluster using a separate account.

  ```
  ssh USERNAME@teach.scinet.utoronto.ca
  ```

  Your USERNAME for the Teach cluster will be of the form `lcl_uotphy1610s...`

  You will receive information regarding your USERNAME and password later in the course.
- Initially, you can choose to do the assignments on your own computer, provided it has a unix-like environment with the g++ compiler, make, and git.
- If you want to keep working on SciNet after the course, get an Alliance/SciNet account, See www.scinet.utoronto.ca/getting-a-scinet-account

Assignments and grading

- **10 programming assignments** (so nearly weekly) will be posted on the website.
- These assignments are **due the next week.**
- They need to be uploaded to the course's website.
- Each student should hand in their own work.
- Assignments are graded on how they can be compiled and run on the Teach cluster.
- The average of the 10 assignments will make up your grade.
  (no midterm nor a final exam)
- All assignments need to be handed in for a passing grade.

Late penalty policy

- Assignments may be handed in up to 1 week after the due date, at a penalty of 5% per day.

  Deviations of this rule will only be considered, on a case-by-case basis, in exceptional circumstances (*i.e.*, **not** "I was busy").
- If, due to exceptional circumstances, an assignment was missed, a make-up assignment on a topic of the instructor's choice can be given at the end of the course.
- Have an accommodation? Email me, please.

Lectures, office hours, questions

Lectures

Lectures will be held in person on Tuesdays and Thursdays from 11:00 AM to 12:00 noon (EST) in the SciNet Teaching Room, which is located on the 11th floor of the West Tower of the MaRS building, 661 University Ave., Suite 1140, Toronto, ON M5G 1M1.

Lectures are recorded and posted on the site afterwards (often towards the end of the day).

Office hours

- In Person, Wednesdays from 11:00 am to 12:00 pm, in the SciNet Teaching Room, starting next week.
- Virtually over Zoom, Fridays from 12 noon to 1 pm.

Questions/comments/concerns/etc. about the course?

Add a discussion topic on the forum on the course website or email courses@scinet.utoronto.ca.

Course Outline

1. Software development

- C++
- Modular programming
- Building software with make
- Arrays and objects
- Version control with git
- Unit testing
- I/O

2. Numerical tools

- Using libraries
- Ordinary differential equations
- Partial differental equations and linear algebra
- Fast Fourier transforms
- Random numbers/Monte Carlo

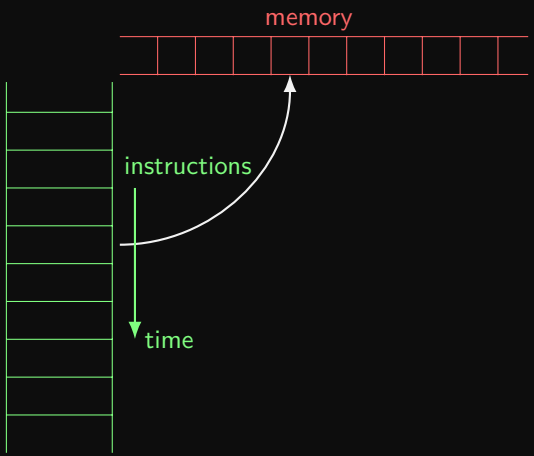
3. High-performance

- Profiling tools
- Intro to parallel computing
- Batch processing
- Shared memory programming
- Distributed parallel programming
- GPU programming

Scientific Software Development

Basic programming concepts

Basic programming concepts



- **Von Neuman model:** A computer executes a set of instructions one-by-one on values stored in memory.
- A program contains a set of instructions.
- When a program is running, its instructions and its data are held in the computer's memory. The data in memory is also called its state.
- Each instruction will have a net effect on the program's state.
- There is limited set of predefined instructions, in terms of which we must express all other actions.

Programming concepts: Programs and functions

- An **algorithm** is a common pattern of actions to achieve a specific net effect (*computation*). Algorithms are described in words and math, and in a specific programming language.
- A **function**, **procedure**, or **subroutine** is a set of actions, written in a specific programming language, that define a new action.
- A **program** is a function that can be executed (a.k.a. application or app).

```
#include <iostream>
int main() {
  std::cout << "pi=" << computepi(1e-12) << "\n";
}
```

Ramanujan's formula for π:

$$\frac{1}{\pi} = \sum_{k=0}^{\infty} \frac{2\sqrt{2}}{99^2} \frac{(4k)!}{k!^4} \frac{26390k + 1103}{396^{4k}}$$

```
#include <cmath>
double computeterm(int k) {
    return 2*sqrt(2)/pow(99, 2)
          *tgamma(4*k + 1)/pow(tgamma(k + 1), 4)
          *(26390*k + 1103)/pow(396, 4*k);
}
double computepi(double accuracy) {
    double sum = 0;
    for (int k = 0; ; k++) {
        double term = computeterm(k);
        if (term < accuracy) break;
        sum += term;
    }
    return 1/sum;
}
```

Programming concepts: Programs and functions

- An **algorithm** is a common pattern of actions to achieve a specific net effect (*computation*). Algorithms are described in words and math, and in a specific programming language.
- A **function**, **procedure**, or **subroutine** is a set of actions, written in a specific programming language, that define a new action.
- A **program** is a function that can be executed (a.k.a. application or app).

```
def main():
    print("pi = ",computepi(1.e-12))
if __name__ == "__main__":
    main()
```

Ramanujan's formula for π:

$$\frac{1}{\pi} = \sum_{k=0}^{\infty} \frac{2\sqrt{2}}{99^2} \frac{(4k)!}{k!^4} \frac{26390k + 1103}{396^{4k}}$$

```
from math import sqrt,factorial
def computeterm(k):
   return 2*sqrt(2)/99**2*(
          factorial(4*k)/factorial(k)**4
          *(26390*k + 1103)/396**(4*k))
def computepi(accuracy):
   sum=0.0
   k=0
   while True:
      term = computeterm(k)
      if term < accuracy:
         break
      sum += term
      k += 1
  return 1/sum
```

Programming concepts: Languages

- The computer's Central Processing Unit (CPU) does not understand programming languages, only **machine code**.
- To execute code written in a programming language, one needs another program, either a
  - **Compiler**: translates source code files into **executable** or **object** files containing machine code.
  - **Interpreter**: does that translation on the fly, one line of code at a time.
- C++ falls in the category of compiled programming languages.
- Python is an example of an interpreted language.

Programming concepts: State

- Program state is stored in memory.
- At least part of the state is made up of the program's **variables**.
- Variables are stored values that are assigned to a **variable name**.
- This variable name is associated with a portion of **memory** that holds the variable's value.
- What the variable stores can change in time.

Note on persistance

- The common definition of state above involves only what is in memory.
- When a program ends, its state is gone.
- Files are a way to store data persistently, but fall under **I/O** (input/output)

Programming concepts: Control structures

- Some actions could be done conditionally on the state of the program and external input.
- **Conditional control structures** perform a different actions depending on whether a certain assertion of the state of the system is true.
- These are usually some variation of an `if-then-else` statement.
- Repetition of a set of actions, *i.e.*, **loops**, are also a type of control structure: they keep doing the same while there are loop iterations left.

Programming concepts: I/O

Programs can receive input

- Interactive (keyboard, mouse, camera, microphone)
- Files containing parameters
- Files containing data
- Input from other programs
- Input from a local network or from the internet

Programs can (should) produce output.

- Output to console
- Graphical output
- Output to files
- Output to other programs
- Response to web requests

Other programming paradigms

This overview described so-called imperative programming paradigm, in which a list of commands acting on data is executed in order.

There are also other paradigms:

- functional programming
- declarative programming
- object-oriented programming
- generic or metaprogramming.

Imperative programming mimics more or less what the computer actually does when running a program, and will be our main focus.

C++

Why C++?

We'll be using the C++ language in this course.

It's not the simplest language, but it is a language that can cover all use cases in this course.

Advantages

- High performance
- Both low-level and high-level programming
- Ubiquitous and standardized
- Useful libraries
- Modular design
- Supports imperative, functional, object-oriented, and metaprogramming
- Supports many parallelization techniques
- Interoperable with C and Fortran.

Disadvantages

- Precise syntax
- Errors can be hard to interpret
- Non-interactive
- Steeper learning curve
- No standard portable graphics
- Susceptible to hidden performance pitfalls
- Susceptible to memory errors

Note: Fortran shares many of the advantages.

C++ Introduction

- C++ is a **compiled** language: files containing the basic 'actions' are to be compiled into a set of basic machine language instructions that the CPU can execute.
- The C language, upon which C++ builds, was designed for system programming.
- The C language has a very small base.
- Most functionality is in (standard) libraries.
- Every 3 years, a new C++ standard comes out, which is mostly backwards compatible.
- For definiteness sake, use the **C++17** standard.

C++ Introduction: Basic C++ programming

The following code prints "Hello, world!" on the console:

```
// @file helloworld.cpp
// Hello world program in C++
#include <iostream>
using std::cout;

int main()
{
   cout << "Hello, world!\n";
}
```

To run this, we need to compile the code.

1. When we will do this on the teach cluster:

```
$ ssh USERNAME@teach.scinet.utoronto.ca
```

2. First, avail yourself of a g++ compiler:

```
$ module load gcc
```

3. Start a new code file in a text editor, *e.g.*

```
$ nano helloworld.cpp
```

4. Type in the code, save it, and exit the editor.
5. Then, compile this into an executable

```
$ g++ -std=c++17 -o helloworld helloworld.cpp
```

6. Finally, run it.

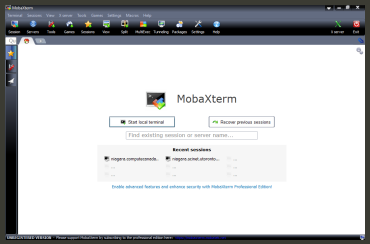
```
$ ./helloworld
Hello, world!
```

Short intro to the terminal a.k.a. console

How to get a terminal

On Windows

Get MobaXterm:

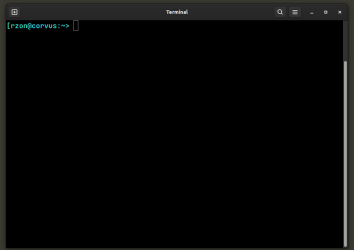


MobaXterm's local terminal runs the bash shell and comes with ssh and X11.

You can also use the Linux Subsystem for Windows.

On Linux

Find your terminal application.



The most common shell interpreter on Linux is bash.

It should have the ssh command.

On MacOS

Find your terminal application.

The default shell is zsh (similar to bash).

It should have the ssh command.

Command line interface

Command prompt

There is a prompt, *e.g.* `"[rzon@teach01:~]$"` after which you can type in commmands.

Any command you type at the prompt is read by a shell interpreter. Teach uses the bash shell.

Commands are either:

- built-in, or
- provided by executables in standard locations (encoded in the so called PATH variable), or
- executables of which the path is specified

Current directory

You are always "in" a current directory/folder in the file system tree. Your default directory, called your "home" directory, is where you start.

You can change to a directory with `cd DIRNAME`

`~` is a shorthand for that home directory.
`.` is a shorthand for the current directory
`..` is a shorthand for the parent directory.

Command examples:

- List the files in the current directory with `ls`.
- If the current directory contains an executable "first", execute it with the command `./first`.
- Connect to a different computer with `ssh`.

Command line arguments

After a command, more words can be entered, the "arguments" of the command.