P6 – Scientific Programming

Marcus Mohr
Jens Oeser

Geophysics Section
Department of Earth and Environmental Sciences
Ludwig-Maximilians-Universität München

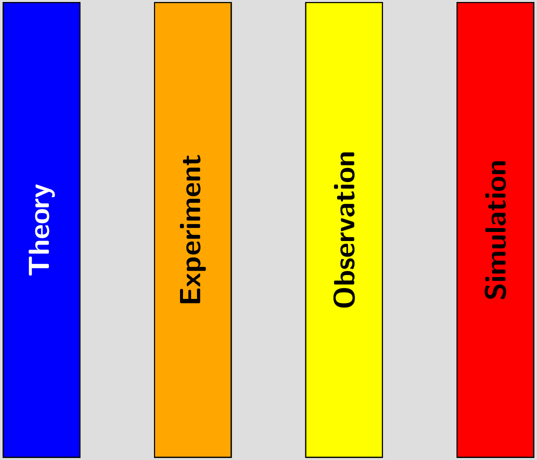
SoSe 2021

Part #1

Motivation

why program? computational science?

Pillars of Scientific Discovery



Why Perform Simulations?

- Theoretical description of mantle convection:

$$\text{div } u = 0$$

$$\text{div}\left[\nu\left(\text{grad } u + (\text{grad } u)^T\right)\right] - \text{grad } p + \varrho\alpha\,(T - T_0)\,g = 0$$

$$\varrho\, c_p\left(\frac{\partial T}{\partial t} + u \cdot \text{grad } T\right) - \text{div}\,(\kappa \text{ grad } T) - \varrho H = 0$$

- What happens, if viscosity ν varies?

Why Perform Simulations?

- Theoretical description of mantle convection:

$$\operatorname{div}\ u = 0$$

$$\operatorname{div}\left[\nu\left(\operatorname{grad}\ u + (\operatorname{grad}\ u)^T\right)\right] - \operatorname{grad}\ p + \varrho\alpha\,(T - T_0)\,g = 0$$

$$\varrho\, c_p\left(\frac{\partial T}{\partial t} + u \cdot \operatorname{grad}\ T\right) - \operatorname{div}\,(\kappa\ \operatorname{grad}\ T) - \varrho H = 0$$

- What happens, if viscosity ν varies?

$\Longrightarrow$ Theoretical models alone may not give sufficient insight

Why Perform Simulations? (cont.)

Experiments/Observations may be . . .

. . . too expensive

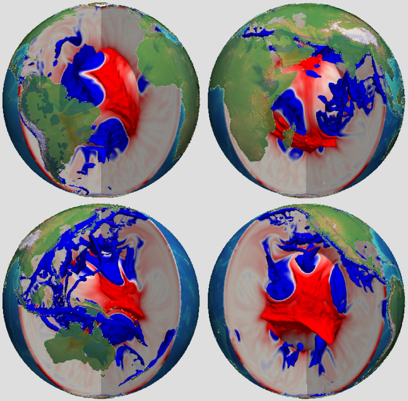
. . . too dangerous

. . . unfeasible

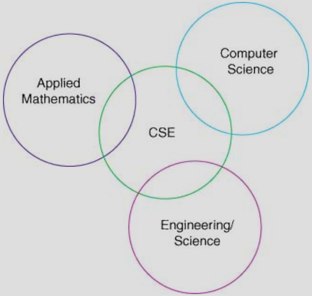
. . . unethical

Ingredients of Simulation

- a mathematical model to *describe* the physical reality
- a numerical algorithm to *evaluate* the model
- a computer program that *implements* the numerical algorithm
- some computer hardware to *execute* the program on
- some means to *evaluate* the results (e.g. visualisation)



Computational Science (from Wikipedia)



Computational science (or scientific computing) is the field of study concerned with constructing mathematical models and numerical solution techniques and using computers to analyze and solve scientific, social scientific and engineering problems.

In practical use, it is typically the application of computer simulation and other forms of computation to problems in various scientific disciplines.

Why Scientist (sometimes) need to Program

- While a multitude of computer programs exist for simulating stuff, none may include the model/feature you are interested in
  "Where can I buy that mantle convection code that implements my brand-new mineral physics model of temperature/pressure dependent viscosity?"
- Thus, you may need to implement your own code, or extend an existing one.
- Writing a script for Matlab, Maple, Gocad or some other tool is also a form of programming.

SuperMUC

percentage of total CPU cycles awarded to all projects

split by project using language

Fortran	72%
C++	54%
C	41%

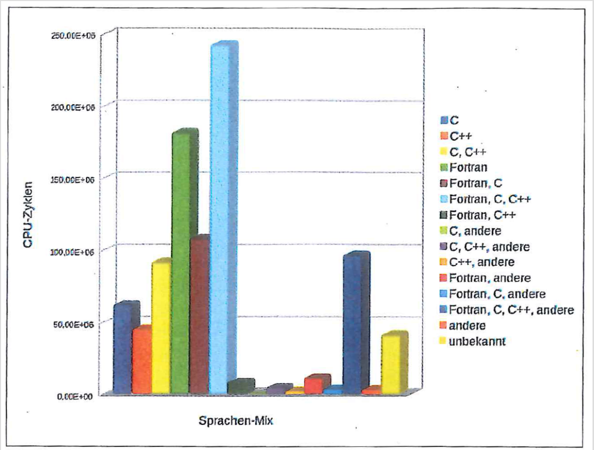


Abbildung 1: Anteile der Programmiersprachen an aktiven Projekten, die im ersten Halbjahr 207 auf dem Super MUC liefen.

(LinuxMagazin 12/2017)

Why C?

1. C is a modern versatile language used in many fields.
2. A lot of scientific code (for number crunching/HPC) today is being developped in C++.
3. While C is not a 100% subset of C++, it nearly is.
4. The syntax of C is basically identical to that of C++ and Java (minus additional constructs in those languages)
5. Language still has fewer features to confuse students than C++.
6. Language demonstrates limits and restrictions of programming.

Why not C++? [Citing from Scott Meyers: Effective C++, 2009]

The easiest way is to view C++ not as a single language but as a federation of related lanuages. Within a particular sublanguage, the rules tend to be simple, straightforward, and easy to remember. When you move from one sublanguage to another, however, the rules may change. To make sense of C++, you have to recognize its primary sublanguages:

- **C:** Way down deep, C++ is still based on C. The rules for effective programming reflect C's more limited scope: no templates, no exceptions, no overloading, etc.
- **Object-Oriented C++:** This part of C++ is what C with Classes was all about: classes (including constructors and destructors), encapsulation, inheritance, polymorphism, virtual functions (dynamic binding). etc. This is the part of C++ to which the classic rules for object-oriented design directly apply.
- **Template C++:** This is the generic programming part of C++. In fact, templates are so powerful, they give rise to a complete new programming paradigm, template metaprogramming (TMP).
- **The STL:** The STL is a very special template library. Its conventions regarding containers, iterators, algorithms and function objects mesh beautifully. The STL has a particular way of doing things you need to be sure to follow.
- **Functional Programming:** new since C++11 (missing in original quote)

Other Alternatives?

- FORTRAN77
  (student prejudices, confusion with modern Fortran, no true FORTRAN77 compilers available any more, we do new stuff mostly in Fortran or C++)
- Fortran{90,95,2003,2008}
  (module induced compilation dependencies, compiler support for specific features, assumed shape arrays vs. automatic arrays)
- Python
  (hugely popular and definitely nice, but neither compiled nor an HPC language)