Object Oriented Scientific Programming with C++ (wi4771tu)

Motivation

MATLAB, Python & Co. are very good tools for developing prototypes and for testing new scientific ideas 'within a minute'. Their mathematical notation and the wealth of excellent add-ons addressing all types of mathematical problems make it easy to implement numerical algorithms without struggling with technical details. For instance, the solution of a linear system of equations in MATLAB is as easy as typing `x=A\b`.

The downside of this programmers' comfort zone is threefold:

- Firstly, you hardly know what your code is doing 'under the hood' so that you have to trust the add-on providers that their routines are implemented correctly. Even then you cannot be totally sure that you are using these toolboxes in the correct way and they won't tell you.
- Secondly, it is wishful thinking that you can exploit the full power of modern computers by naively calling some 'parallelised' or 'CUDA-enabled' black-box routines. Doing so, you can be lucky if the code still produces correct results and is not slower than before.
- Thirdly, if you start using low-level programming techniques such as loops, branching, etc. then the nature of interpreted programming language strikes back making your code impractically slow.

If you already encountered one or more of these problems then it is time to learn scientific programming in a programming language like C, C++ or Fortran, which allow you to express your algorithms in low-level but still human-readable notation and transform it into hardware-optimised machine code using a compiler. Starting with the right choice of development tools, solving a linear system of equations in C++ is as easy as typing

```
SparseMatrix<double> A(n,n);
Vector<double> x(n), b(n);
ConjugateGradient<SparseMatrix<double> > cg;
cg.compute(A)
x=cg.solve(b);
```

But this time, you have full control over all internals (here: use of a sparse matrix with double-precision floating point data and an iterative conjugate gradient solver) and can adjust them to your particular needs (like the absolute tolerance).

Mission of the course

This course is not a typical C++ programming course that aims at discussing all aspects of the programming language at length, including legacy techniques that lead to hardly-readable codes. It is about object-oriented design concepts and practical aspects of modern scientific programming thereby using C++ as a vehicle only.

This course follows a pragmatic problem-oriented approach. Starting from a concrete problem description like *'How to design and develop an efficient linear solver for Ax=b?'*, only those aspects of the programming language will be introduced and discussed that are really needed to master the problem at hand or make its implementation much more

flexible and efficient. We explicitly focus on the recent language standard C++11, which introduced several very powerful and easy to use concepts for modern scientific programming and overcomes many of the drawbacks and pitfalls of previous standards.

The second aim of this course is to demonstrate a possible workflow for efficient code development using established tools for editing, refactoring, compiling, debugging, and profiling the code. Scientific codes for practical use are quickly becoming too complex to edit them in a standard text editor, compile the source code 'by hand' manually figuring out the inter-file dependencies and identify bugs and efficiency bottlenecks just from visual inspection. We will introduce and utilise several software tools like CodeLite and CMake (possibly also Valgrind if time permits) that simplify these tasks.

Audience and prerequisites

The course is meant for master students from disciplines such as (but not limited to) Applied Mathematics, Aerospace Engineering, Civil Engineering and Geosciences, who are interested in learning modern scientific programming in C++, e.g., as preparation for a master project work that involves a significant programming part. Rudimentary background in linear algebra and, possibly, numerical mathematics is required to follow the problem-oriented teaching approach. Furthermore, good knowledge of at least one programming language like MATLAB, Python, C/C++ or Java is mandatory since this course is not meant as an introduction to the basics of programming.

Course schedule

The course consists of a weekly lecture (Tuesdays 10:30-12:30 in EWI-lecture hall J) and a mandatory weekly lab session (Fridays 13:30-17:30 in DW-PC4 (first floor).

ECTS points

Successful participation will be assessed by practical assignments for each session (1/3 of the final grade) and a programming project (2/3 of the final grade) that can and should be done in groups of 2-3 students. The course gives 3 ECTS.

Further information

For further information about this course please check the TU Delft Course browser
http://www.studiegids.tudelft.nl/a101_displayCourse.do?course_id=41034
or contact Dr. Matthias Möller (m.moller@tudelft.nl).