Scientific Programming

C.W.J. Lemmens, H.X. Lin, C. Vuik
Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

January 20, 2015

1 Course description

This course tries to bring students to a level where they are able to change algorithms from e.g. numerical analysis into efficient and robust programs that run on a simple computer. Important aspects are :

- Learning how to program in a high level programming language.
- Transition from scientific model to a structured program.
- Optimisation, Debugging and Profiling of these programs.

This course mainly concentrates on sequential programming. More advanced topics like threads or parallel (MPI / GPU) programming on supercomputers are the focus of this course.

2 Registration

The lectures and lab sessions will start in February 10, 2014. If you intend to take the course, please register by sending an email to h.x.lin@tudelft.nl with the subject 'Registration for Wi4260'. Due to the limited capacity of the computer lab there is a maximum of the number of participants.

3 ECTS

This course has the equivalent of 3 ECTS points.

4 Responsible Instructors

Name	E-mail
Ir. C.W.J. Lemmens	C.W.J.Lemmens@tudelft.nl
Prof.dr.ir. H.X. Lin	H.X.Lin@tudelft.nl
Prof.dr.ir. C. Vuik	C.Vuik@tudelft.nl

5 Schedule

Lectures: Tuesday 13:45 - 15:30, Lecture hall J (Building EEMCS, Mekelweg 4), Period: from February 10 to March 24, 2015

Lab sessions: Thursday afternoon, 13:30 - 15:30, Room 05.140 (5th floor, Building EEMCS, Mekelweg 4), Period: From February 19 to April 2, 2014

6 Course Language

The course will be given in English.

7 Course Material

Writing Scientific Software - Suely Oliveira & David Stewart. Cambridge University Press, ISBN-13: 9780521675956 (ISBN-10: 0-521-67595-2)

8 Course Contents

The contents of lectures 1 to 7 follows hereafter :

8.1 Introduction to programming in general

- Numerical software introduction : possible problems
  - Integer numbers and their limitations
  - IEEE floating point numbers and their limitations
  - Priorities for algorithms and software
  - Famous disasters
- Software priorities
  - Correctness
  - Numerical stability

  - Flexibility
  - Efficiency
- Developing software
  - Procedures, Functions, Variables
  - Overview of languages (C, Fortran, Java, C++, Python)
  - Compilers, linkers, loaders
  - Some simple examples (e.g. Simpson integration, approximate π ...)

This is roughly covered by the chapters 1-5 plus chapter 6 until 6.4

8.2 Software design

- Software lifecycle (design, testing, debugging, use, redesign ad inf.)
- Large scale considerations
  - Problem analysis : what needs to be done
  - How to cooperate with other developers
  - How to coordinate development : what interfaces do we need
  - Software portability : on which platforms should it run
  - Language(s) to use
  - Main data structures to be used
  - Software libraries to be used
- Medium scale considerations
  - Organizing functions and files
  - Detailed specification of interfaces and data structures
  - Comments and documentation
  - Cross language development (call by reference vs by value)
  - Software layers
- Small scale considerations
  - Memory usage and data locality
  - Avoiding premature optimisation
  - Zero and one based relative indexing
  - Variables and function names
  - Style and layout (indentation, upper and lowercase names), role of editors in achieving this

This is all roughly covered by chapter 7

8.3 Data structures

- Data structures (structs, records and classes)
- Global variables (and how to avoid them)
- Multidimensional arrays
- Using pointers to arrays and functions

This is covered by chapter 8

8.4 Testing, debugging and profiling

- Debugging : how to use a (commandline) debugger
- Profiling : how to use a profiler

This is mostly covered by chapter 9

8.5 Efficiency in both computing time and memory usage

- Choosing the right algorithm
- Caches and memory hierarchies
- Virtual memory and paging
- Pipelining and loop-unrolling
- Indexing versus pointers for dynamic data structures

This is mostly covered by chapter 11 and 12 (except Blas/Lapack)

8.6 Optimisation and dynamic memory allocation

- Optimisation, both by the programmer and using automated optimisers
- Memory management : allocation and deallocation
- Memory bugs and leaks: some horror examples : (e.g. int array[10]; array[10] = x;)
- Memory leak checkers (`MALLOC_CHECK_` , valgrind)
- Debugging tools

This is about chapter 13, 14 and 15

9 Scientific software sources and libraries

- How to use Blas and Lapack from C
- How to use Fftw (Fast Fourier Transforms)
- How to use Eigen2/3 and Meschach
- Basic Unix commands (make, diff, rsync, grep, find) ?

These are the topics described in chapters 16 and 17 (part IV : Tools)

10 Some loose ends

10.1 Other possible topics

- Short introduction to C programming
- Short introduction to object oriented programming (using Java ?)
- Using an IDE like Eclipse ?

11 Excercises

Excercises will be taken from chapter 5, 10 and possibly from 18 and 19

12 Study Goals

The goals of this course are described below. After this course the student is able to :

- Use structured programming : he/she knows how to go from a mathematical model via pseudo code to a working C-program
- Write programs that are fault tolerant (error checking)
- Use debugging tecniques and tools : the student is able to find out what could be wrong with a program if it doesn't work as expected : discretisation errors, round off errors, memory allocation errors, memory leaks, memory corruption...
- Analyse a program using timers and profilers in order to optimise the performance
- Use the most commonly used scientific software libraries, like Blas, Fftw and Meschach
- Use basic (Unix) commands to inspect source code and/or data files and or to move data around between different locations

13 Education Method

Lectures, combined with practical work in a labroom. Theory and exercises will be mixed during these sessions. Further there will be larger lab sessions purely dedicated to exercises and "hands on".

14 Assessment

The final assessment is combined written exam + lab exam (3 hours).

15 Remarks

This course should be an excellent preparation for the courses on Parallel Programming both for MPI and GPU and is mandatory for doing a master thesis in the Numerical Mathematics group. It should bridge the gap between numerical analysis and real scientific computing.