Design and Computing 1

Introduction to Scientific Computing

Daniel Poole
Department of Aerospace Engineering
University of Bristol
d.j.poole@bristol.ac.uk

2017

LECTURE 1

TODAY

- Introduction to computing
- What is scientific computing?
- Why this course?

What is a Computer?

- Albert Einstein:
  "Computers are incredibly fast, accurate and stupid. Humans are incredibly slow, inaccurate and brilliant. Together they are powerful beyond imagination."

- c.1830: Invented by Charles Babbage - mechanical system!

- 1936: Alan Turing proposed a machine that could store a set of instructions (a program)

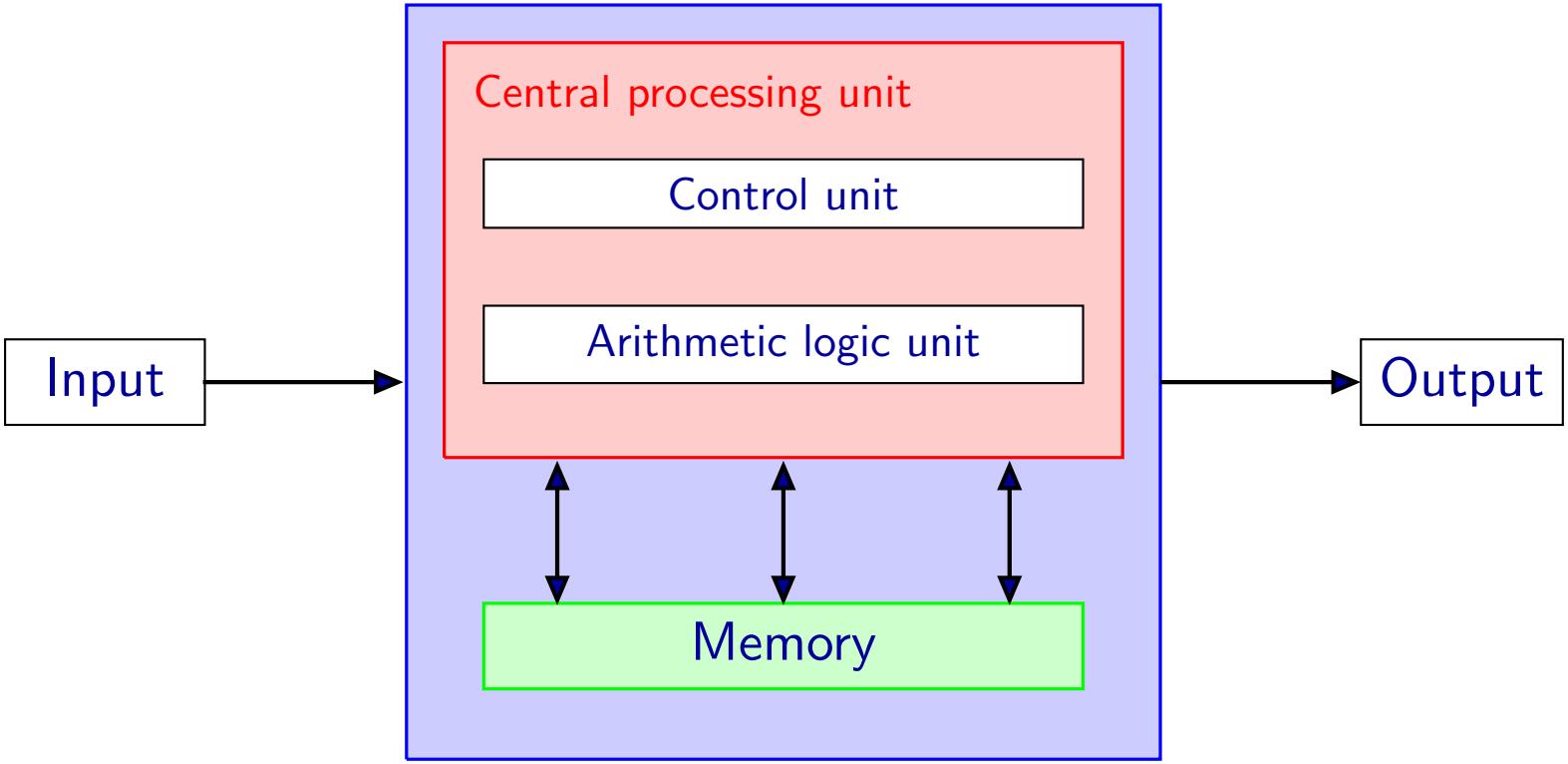
- 1945: John von Neumann describes the architecture that is now commonplace

- Computing power has increased remarkably in the last three decades

- Computers will only do what programmers tell them to do, and how they tell them to do it

What is a Computer?

Computer consists of hardware, software and operating system. The 'machine':



An Operating System (OS) is the mechanism to access the computer's hardware - this is the 'environment'. A programming language is the language used to define a set of commands/tasks the programmer wishes the computer to perform. This course is aimed at training you in the essentials of programming. These skills are independent of the operating system.

What is a Computer?

There are many options for the OS. If you are a '**consumer**' of data (majority of computer users) normally use the most easily-accessible technology. Windows OS and Office on simple Intel hardware dominate private market due to ease of use and not needing to do anything clever.

If you are a '**producer**' or 'user/analyser' of data (all numerical and computational scientists) you need a more robust and efficient environment. Research normally performed on Linux machines.

To get the computer to perform an 'operation':

1. A source 'code' is developed (a text file) in a high level programming language

2. A 'compiler' is used to convert the code to a list of commands in machine code

3. This creates a 'binary' or 'executable' that will run on the local operating system.

Computer Development

In 1965, Intel co-founder Gordon Moore published an article where he predicted that the number of components (e.g. a transistor, resistor, diode or capacitor) would double every 12 months due to improvements in design and manufacture. This became what is now known as "*Moores law*". Top performing computers:

CPU development is levelling off; primarily power issues and silicon manipulation limits. Machine power is now rising rapidly due to increasing CPUs/cores.

Flop: floating point operation

Looking to be in the exascale (10^{18} Flops/s) by 2020!

Computer Development (Bristol)

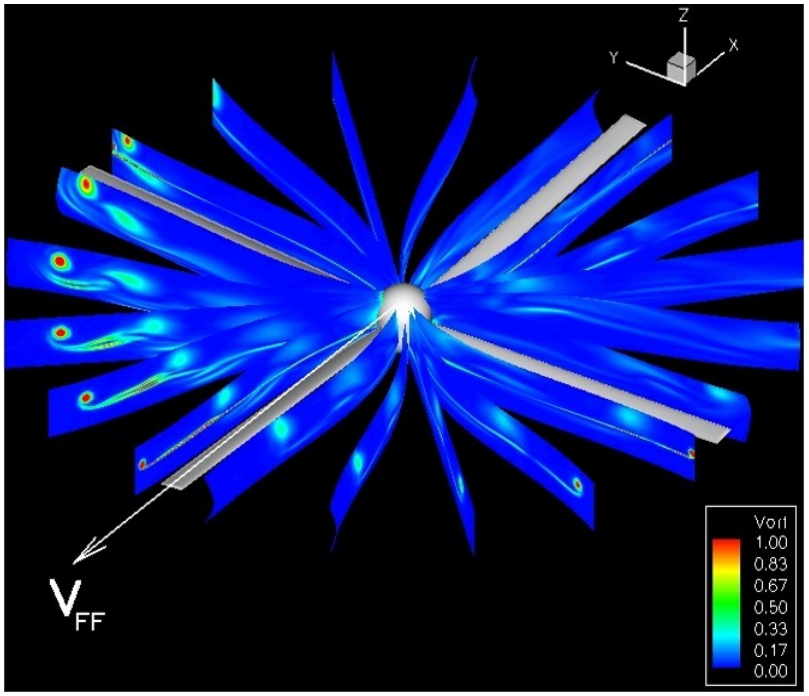
Bristol University management understand the need for high performance computing (HPC). Up until the early-2000s, most HPC resource was national but this causes bottlenecks so Bristol began investing in HPC. Now, over 16million invested in HPC at Bristol in last 10 years and we now have over 1000 users in a wide variety of departments at the university The 'old' machine room is above physics:



The university has recently opened a new phase (phase 4) of the Blue Crystal cluster with 14,000 'cores' and 32 dual-GPU servers (capable of 800TFlop/s)

Scientific Computing

One of the most demanding research areas in terms of computer requirements is Computational Fluid Dynamics (CFD). This is the numerical simulation of fluid flows and is used extensively in all areas of aerodynamics and fluid dynamics. All cutting-edge research in this field is done by researchers creating their own codes within a Linux environment to be able to exploit the available hardware.



Prof. Allen's flow solver. Example contains 32 million cells. Need to split the domain down and run on more processors. Parallel code developed to split domain into smaller tasks.

1 core	over 2 years
256 cores	3 days

Scientific Computing

Other related areas:

- Climate modelling: Also a CFD-type code, but much larger length scales, and less accuracy required; gross trends required for weather. Met Office has largest computer system in UK (over 100,000 CPUs, running Linux).
- Drug design: Numerous research groups modelling molecular interactions. Optimum Influenza vaccine simulated at Bristol 2013. Required 1000 CPUs for 27 weeks.
- Genome sequencing: Very simple computations, but trillions of combinations must be analysed.
- Google?: Huge Linux servers with parallel filesystems.
- Video games: Source code developed with graphics links and exploitation of graphics cards; driven technology development.
- Mobile phones: Apps coded, many by amateur code developers, using Java.

Course Aims

Every area of science now requires exploitation of powerful hardware available. There are no commercial or freely-available tools that automatically exploit available power. You must at least be aware of the basics of structured code development. **This is as fundamental in science as mathematics.**

The primary aim of this course is not to make you good at programming in a specific programming language, but to help you understand the thought processes behind taking a problem and programming a computer to find a solution. Obviously, to learn the art of programming, we must learn a specific programming language (see slides below), but it must be emphasised that we are learning the process of programming and that this process can be ported to any other programming language you may come across during your careers.

Programming is an exercise in LOGICAL THINKING. This course will help you develop a logical way of approaching and tackling a given problem; an invaluable skill to have. It is therefore worth noting that a 'difficult' programming exercise is rarely difficult, it is just complicated.

A good programmer will be able to solve almost any problem, assuming it can be represented as a series of 1's and 0's with some input and some output.

The "Engineering Method"

- e.g. Describe the flow structure around a hovering rotor?

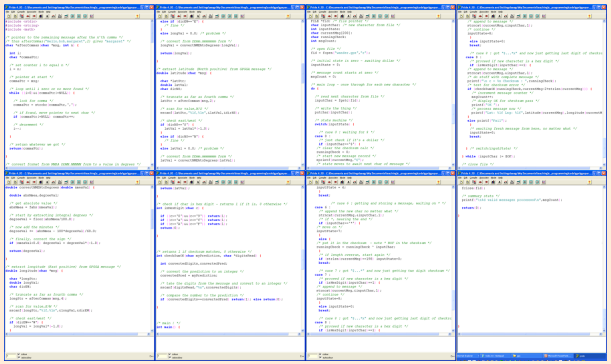
PROBLEM

Flow around rotor?

$\Rightarrow$

SOLVE

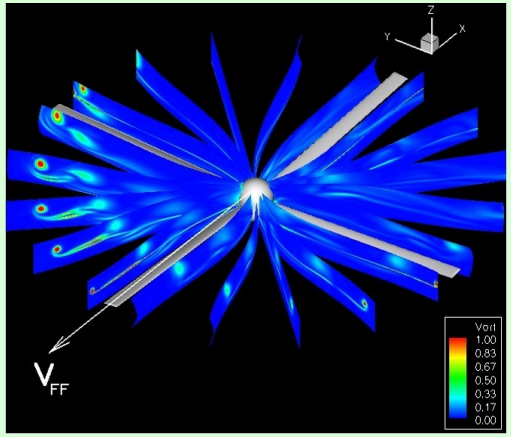
Programme computer to solve complex flow equations



$\Rightarrow$

REPORT

Scientific plotting software



A Computer Program

A program is a series of commands that manipulates some input (numbers) to produce some output (some other numbers). Analogous with the engineering method:

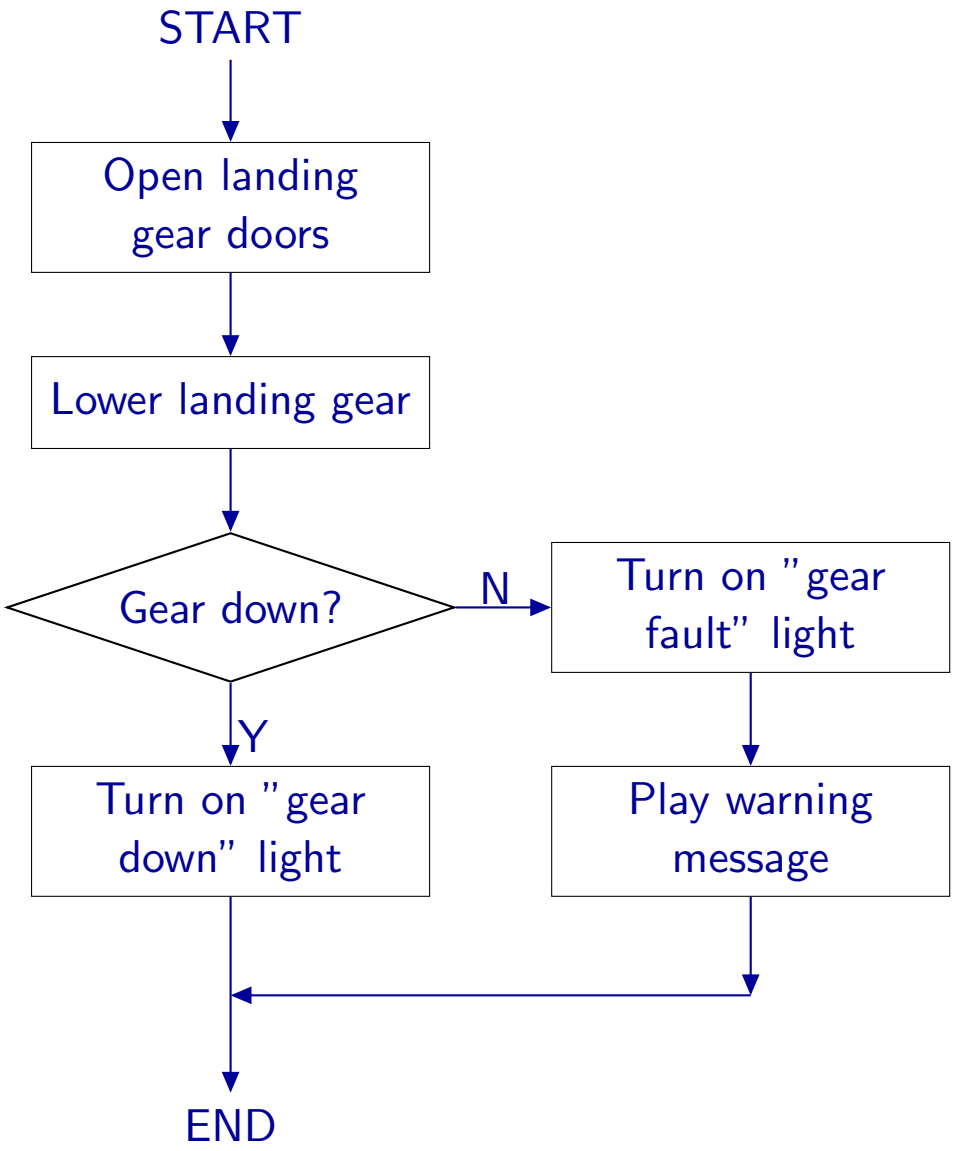
$$\text{PROBLEM} \Rightarrow \text{SOLVE} \Rightarrow \text{REPORT}$$

$$\text{Input} \Rightarrow \text{Program} \Rightarrow \text{Output}$$

A computer programmer takes a problem and breaks it down into a system of logical steps, which can then be input into a computer to solve the problem being asked.

A Computer Program

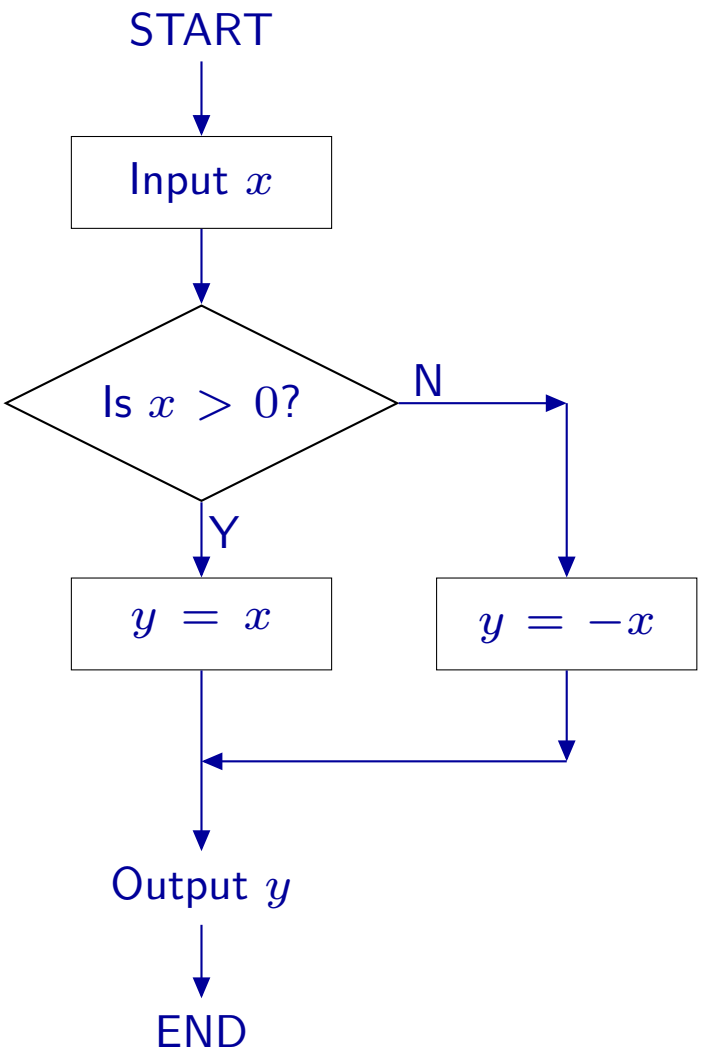
A working example of a system broken down into logical steps. Can be visualised as a flowchart:



A Computer Program

Computers deal with numbers (well, strictly it's 0 & 1), so what about a more mathematical example.

Absolute value:



Logical Thinking

Before progressing, it is worth understanding what a program can and cannot do. This is imperative in being able to take an engineering problem, and solve it.

A program cannot solve a statement. For example:

$$x^3 - x - 2 = 0$$

is a statement of fact, but does not tell us how to find what x is. What we need is a recipe for finding the value of x such that the above statement is true. Throughout the course we will work through examples of taking a given problem and splitting it down into workable algorithms that can be coded. Take, for example, solving the above equation:

```
a=1, b=2
DO
  x=(a+b)/2
  IF(x^3 - x - 2 > 0), b=x
  IF(x^3 - x - 2 < 0), a=x
  IF(x^3 - x - 2 = 0) STOP
REPEAT
```

This is a recipe (a simple linear bisection method) for finding a value of x that satisfies the given statement.

Programming Languages

We need to write our program in a language that the computer can understand. To do this, we write in a programming language, and then either **COMPILE** to create an **EXECUTABLE** (this is called a compiled language), or run through an **INTERPRETER** (this is called an interpreted language).

Each has there own advantages and disadvantages and each are used in numerous quantities in industry and academia, but we will be learning to use one of each:

Compiled	Interpreted
C	MATLAB

Interpreted languages tend to be more compact in nature, but compiled languages tend to run much quicker. For example, most engineering analysis (which often requires lots of calculations) will use a compiled language, normally either C of FORTRAN.

Programming Languages

Example of a simple program to print something to screen:

C

```c
#include <stdio.h>

int main() {
  printf("Hello World!\n");
  return(0);
}
```

MATLAB

```
function example
  'Hello World!'

end
```

NOTE: MATLAB is more of a development package, than a fundamental programming language.

Programming Languages

We will come back to MATLAB in later lectures, but for now we are concentrating on C. This will help you to understand some fundamental programming concepts about what the computer is doing with the data you give it. MATLAB makes a lot of assumptions about data types that can make bugs easy to write into code.

DEMO: compiling C

COURSE OUTLINE

Lectures

One lecture (Monday 10-11am) a week for TB1, broken down as follows:

- Wk 1-7: Intro to programming in C
- Wk 8: Study week (no lecture)
- Wk 9-10: Intro to programming in MATLAB
- Wk 11-12: Scientific documents and other loose ends

Labs and Assessment (TB1)

Programming forms 25% of Design and Computing 1. There are 25 total marks available for the programming exercises and are assessed in class. One computer lab (Tuesday 2-4pm or 4-6pm) a week for TB1. **Groups alternate every week**.

- Wk 2-7: 5 simple examples (in C) based on lectures
  - Each pass/fail – 1 mark per exercise – Completed **by the end of lab in week 7**.
- Wk 8: Study week (no lab)
- Wk 9-10: 2 simple examples (in MATLAB) based on lectures
  - Each pass/fail – 1 mark per exercise – Completed **by the end of lab in week 12**.
- Wk 11-12: 1 more in-depth example (in MATLAB)
  - 3 marks – Completed **by the end of lab in week 12**.

Labs and Assessment (TB2)

TB2 forms a more major programming exercise, but is slightly more complicated (details will be given closer to the time):

- Will assess your ability to be given a problem and split this down into a programmable solution
- Have two consecutive lab sessions (the timetable for this will be released closer to the time)
- The assignment is worth a total of 15 marks

Extra Reading - C

This course is entirely self-contained, but for another point of view on C programming, try:

- Kernighan, B. W. and Ritchie, D. M. "The C Programming Language". Prentice Hall, 1988.

There are also a myriad of online self-teach courses and materials:

- Good all-rounder: `http://c.learncodethehardway.org/book/`
- Easy to use GUI: `http://www.learn-c.org/`
- MIT open courses: `http://ocw.mit.edu/courses/intro-programming/`
- etc.

Extra Reading - MATLAB

This course is entirely self-contained, but for further examples of MATLAB programming try:

- Hahn, B. D. and Valentine, D. T. "Essential MATLAB for engineers and scientists". Butterworth-Heinemann, 2007.

Also the official MATLAB tutorial (you will have to create a Mathworks account to access this)

- `https://uk.mathworks.com/support/learn-with-matlab-tutorials.html`

Summary

- Many of you will only have experienced Windows and Office, plus a web browser. This is simply a data consumption environment.
- We need tools for being able to do things that aren't provided for a data consumer.
- Fundamental code development is a core skill/knowledge in all areas of science (and non-science areas).
- This course will introduce you to the core principles of structured programming
  - Aim is NOT to teach you commercial software development skills!

LAB

- Starts next week
- Before then, go through 'Getting Started' sheet on blackboard which shows you how to compile a basic code on the engineering machines