CS319: Scientific Computing

Week 12: Wrap Up

Dr Niall Madden

9am and **4pm**, 27 March, 2024

Slides and examples:
https://www.niallmadden.ie/2324-CS319

News and Updates

- Lab 6: grades were posted last week. Let me know if you have questions.
- Lab 8: grades will be posted within the next week.
- Presentations: today and tomorrow. See https://www.niallmadden.ie/2324-CS319/2324-CS319-Projects.pdf

Why C++

In CS319, we have mainly used C++. Although it is not so popular in academic settings it (and its older sibling C) remain important languages in industry. This is mainly for 2 reasons

- The are VERY fast, compared with interpreted languages such as Python and R. This is because the compiler is highly optimised.
- They are low-level: since they can directly access memory addresses, they are used for hardware programming.

Why C++

Of course, C and C++ have disadvantages compared to (say) Python.

- The have a steeper learning curve: it is harder to get started.
- No good notebook solution yet.
- Can have security risks (see e.g., report from White House (US) last month which highlighted C and C++ has "non-memory safe languages".
- If you use Python **carefully**, then it can be quite fast.
- Python has many more modules and libraries.

Is there a "best of both worlds"? In a sense, there are several.

- Python is written in C, and so it is possible to integrate C/C++ libraries with it.
- For example, when you use `numpy` in Python, you are using an interface to libraries written in C and C++.
- You can write C++ code that can be run from Python, in one of two ways:
  - Write a Python module in C++ using the `PyBind1` library.
  - Use the `ctypes` module in Python to access a C++ library.
  - Use just-in-time (`jit`)compilation of your Python code.

  The first two of these are not too complicated, but are beyond what we can cover in this module. However, using `jit` is easy...

Older languages, such as C and C++, are "**compiled**": the source code is converted to a machine language before it is run. (Easy to optimised; not really interactive).

Many more modern languages are "**interpreted**", where the code is executed at run-time, usually line-by-line. (Hard to optimise, can be interactive).

The newest languages, such as Julia, try to combine both ideas by compiling only as needed.

In Python, we can do this with the `jit` decoration, which is part of the `numba` module.

```
import numpy as np
import time
from numba import jit
```

Now define a function for multiplying two matrices:

```
def MatMat(A,B):
    N = len(A)
    C = np.zeros((N,N), dtype=A.dtype)
    for i in range(N):
        for j in range(N):
            for k in range(N):
                C[i][j]+=A[i,k]*B[k,j]
    return C
```

Let's test this:

```
N = 400;
A = np.random.rand(N,N)
start = time.time()
C1 = MatMat(A,A)
MatMat_time = time.time() - start
print(f"MatMat for {N}-by-{N} matrix took {MatMat_time
    :.3f} seconds")
```

The output I get from this is

```
MatMat for 400-by-400 matrix took 33.387 seconds
```

However, if you load the `numba` module, we can "docorate" the cell defining the function with `@jit`

This causes Python to call a compiler the first time this function is used.

So, for example, when I try

```
@jit(parallel=True, fastmath=True)
def MatMatjit(A,B):
    N = len(A)
    C = np.zeros((N,N), dtype=A.dtype)
    for i in range(N):
        for j in range(N):
            for k in range(N):
                C[i][j]+=A[i,k]*B[k,j]
    return C
```

And when I run the above code (or similar) first time, I get

```
MatMatjit for 400-by-400 matrix took 0.792 seconds
```

Already this is much faster. But some of that time including compilation. So any subsequent time I run it, I get something like:

```
MatMatjit for 400-by-400 matrix took 0.075 seconds
```

Assessment for CS319

The assessment for CS319 is based on

1. 40% based on lab assignments: 10% for each of Labs 2, 4, 6 and 8.
2. 20% from Class Test.
3. 40% for Project
   (a) Initial Project Plan/Discussion [**5 Marks**]
   (b) Project Proposal [**5 Marks**]
   (c) Presentation [**5 Marks**]
   (d) Project code [**15 Marks**]
   (e) Project report [**10 Marks**]

Deadline for report and code is 5pm, Friday **5** April.

Module review

The topics we have covered (not necessarily in order) are:

(a) Basic I/O (`cin`, `cout`), and manipulators (`endl`, `setw`, ...);

(b) Flow of control and looping (`if`, `for`, `while`, etc.)

(c) Fundamental data-types (`int`, `float`, `double`, `char`, `bool`, ...). Arrays.

(d) `string`, and C-style strings.

(e) Computer representation of numbers (underflow, overflow, machine epsilon, ...)

Module review

(f) Dynamic memory allocation, including of multidimensional arrays

(g) Functions: default parameter values, overloading, functions as arguments to other functions, recursion, pass by reference/value.

(h) Classes, including the `private`, `public` and `friend` access specifiers.

(i) Classes: constructors and destructors, including copy constructors.

(j) `template`s.

Module review

(k) Function and operator overloading (syntax, precedence, implicit/explicit arguments, the `this` pointer, unary/binary operators, assignment operators, ...).

(l) The C++ preprocessor

(m) The Standard Template Library (STL), including containers (especially `set`, `multiset` and `vector`), iterators, algorithms (but not functors); range-based loops.

(n) Optimisation and bisection;

(o) Sorting;

(p) Stacks;

Module review

(q) Numerical Integration;

(r) Solving linear systems by the Jacobi and Gauss-Seidel methods;

(s) Solving diagonal and triangular systems (back-substitution);

(t) Sparse matrix representation, especially triplet and CCS; Matrix-vector multiplication

(u) Reading to and writing from files;

Module review

THE END!!!!!!!!

I hope you have enjoyed CS319, and have learned something (I did!).