Case Study - Matrices

This case study tests the performance of matrix-matrix multiplication, one of the most highly optimised tasks in numerical computation. It compares naive Python and Fortran implementations with using optimised implementations through the Basic Linear Algebra Subprograms (BLAS) in both Fortran and Python. Fortran is used rather than C since simple Fortran is generally faster than simple C for these problems.

For the Fortran examples you will need a Fortran compiler (e.g. gfortran) and an implementation of BLAS. BLAS is available on the SCRTP desktops by loading the modules GCC/7.3.0-2.30 and OpenBLAS/0.3.1

- matrix_python.py

  - A pure Python implementation

  Info:

  This demo by default multiplies together 200 x 200 matrices. Add a number at the command line to do larger matrices

  Try:

  Run the demo (expect ~5-10 seconds), read the code, and work out how it works

  Confirm that the implementation does multiply the matrices together and see if you can think of any way of doing it faster

- matmult_loop.F90

  - A pure Fortran2003 implementation

  Info:

  This exactly replicates the matrix multiplication code from Python in Fortran 2003. By default this will multiply together a 5000x5000 matrix and take about a 1-2 minutes to complete

Try:

Compile with “gfortran matmult_loop.F90 -o matmult”. Then run with “./matmult”. It will tell you how long it takes to multiply the matrices together in ms. Try recompiling with the “-O3” flag to see if it makes any difference. Change the size of the matrix

We know that the code will do about n^3 operations to do the multiplication. Given the clock speed of your computer how does this compare with the maximum possible rate

- matmult.F90

- A pure Fortran2003 implementation using Fortran intrinsic functions

Info:

This does matrix multiplication using the Fortran MATMUL intrinsic function. By default this will multiply together a 5000x5000 matrix and take about a 1-2 minutes to complete

Try:

Compile with “gfortran matmult.F90 -o matmult”. Then run with “./matmult”. It will tell you how long it takes to multiply the matrices together in ms. Try recompiling with the “-O3” flag to see if it makes any difference. Change the size of the matrix

Can you immediately see any reason why our simple loop implementation might be slower than the intrinsics?

- blas.F90

- A pure Fortran2003 implementation using the BLAS library

Info:

This does matrix multiplication using the BLAS library. By default this will multiply together a 10000x10000 matrix and take about about 5-30 seconds to complete

Try:

Compile with "gfortran las.F90 -lblas -o matmult". Then run with "./matmult". If you get errors that talk about "pthread" then also type "-lpthread" on the command line It will tell you how long it takes to multiply the matrices together in ms. Try recompiling with the "-O3" flag to see if it makes any difference. Change the size of the matrix

- matrix_jit.py

- A pure Python implementation using the Numba just in time compiler

Info:

This demo by default multiplies together 200 x 200 matrices. Add a number at the command line to do larger matrices

Try:

Run the demo (expect below a second), read the code, and work out how it works

Compare the speed to the two "simple" Fortran implementations. See that the jitted python is comparable but substantially slower

- matrix.py

- A numpy implementation. Make sure that you load the modern Python implementation module on the SCRTP desktops. They do not all have the same implementation of BLAS behind numpy. To see the version of BLAS being used run the following at the command line

- >>>import numpy as np
  `>>>np.__config__.show()`

Info:

This demo by default multiplies together 10000 x 10000 matrices. Add a number at the command line to do different matrices

Try:

Run the demo (expect a 5-30 seconds), read the code

Compare the speed to the two Fortran BLAS version. It should be very similar