MATH96012 Project 3

Oscar Pring 01198242

November 28, 2019

Question 2

I begin by comparing the relative execution times and the alignment measure, $alpha(t)$, as defined in the description of project 1, for the three simulation implementations. To generate figure 1, the number of simulations, m, the number of particles, n, and the number of time steps, nt, was set to $(m, n, nt) = (200, 100, 1000)$.
Please note, for all computations in question 2, $(s0, A) = (0.2, 0.64)$.

In figure 1, we can clearly see that the Fortran and Python implementations produce very similar alpha outputs, whilst the Fortran+OMP implementation produces an alpha output of a similar shape, but with a peak at a lower time step and more variation in the "plateau" after the initial transient.
From figure 2, as expected, we can see that the Fortran implementation is slightly quicker than the Python implementation, whilst the Fortran+OMP implementation is the fastest by a significant margin. I explain my approach to parallelisation within the comments of $p3.f90$.

In figure 3, I look at how varying the number of simulations, m, affects the execution time of the different implementations. As expected from considering the number of floating point operations carried out by the program, the relationship is linear, although there is an unusual peak for the Fortran implementation where $m = 320$. I take a closer look at this in figure 4.

In figure 5, we can see that the relationship between the number of particles, n, and the execution time is exponential. Unsuprisingly, the rate of growth is smallest for the Fortran+OMP implementation, however, the rate of growth is greater for the Fortran implementation than that of the Python one. I suspect that this is because NumPy handles large arrays more efficiently than a "direct translation" of the Python code into Fortran does.

Figures

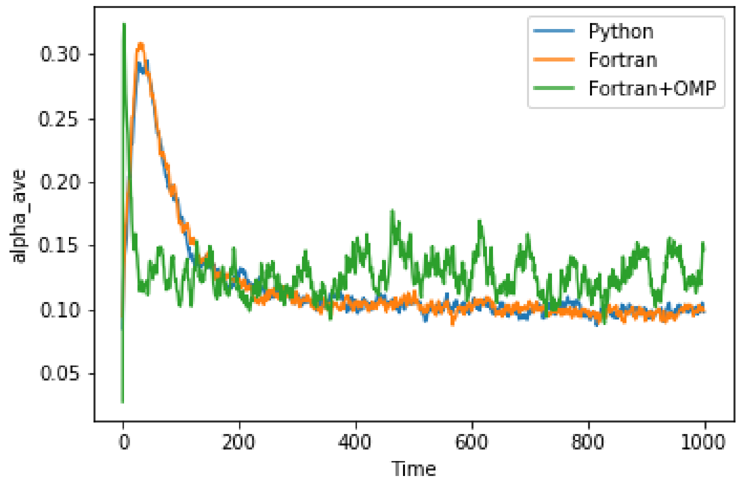

Figure 1: Comparing the alignment measure against time

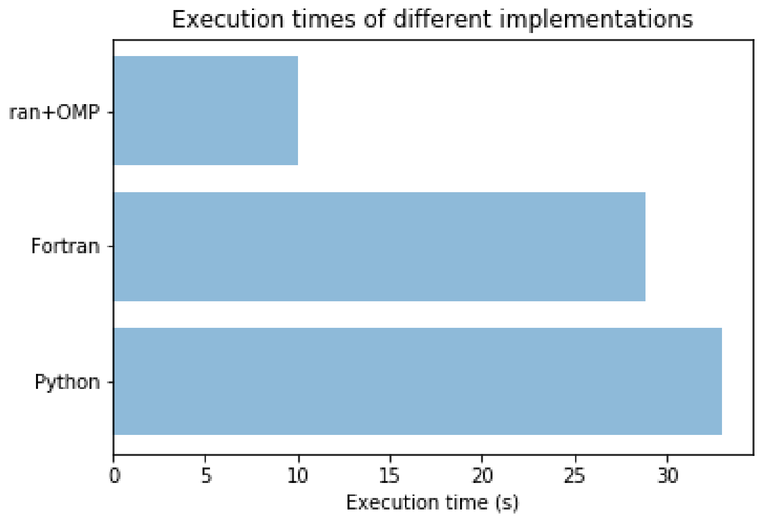

Figure 2: Comparing execution times for the different implementations. Here $(m, n, nt) = (200, 100, 1000)$

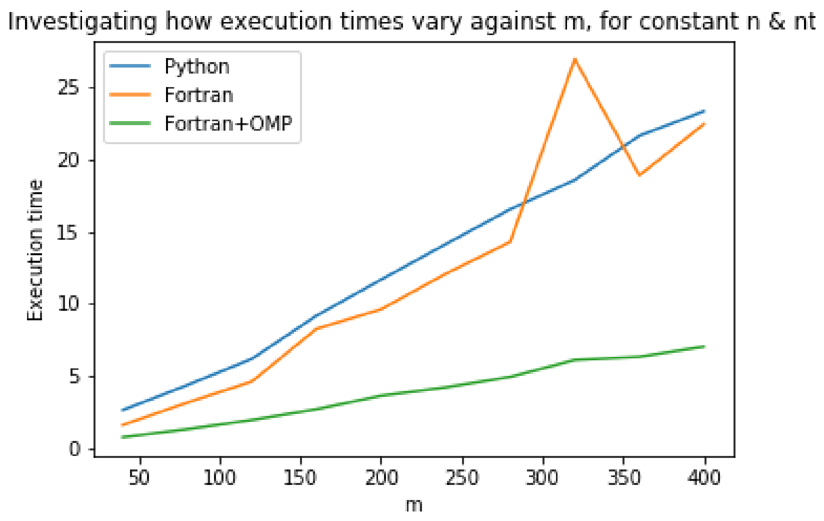

Figure 3: Analysing the effect of m on execution times. Here $(n, nt) = (100, 300)$

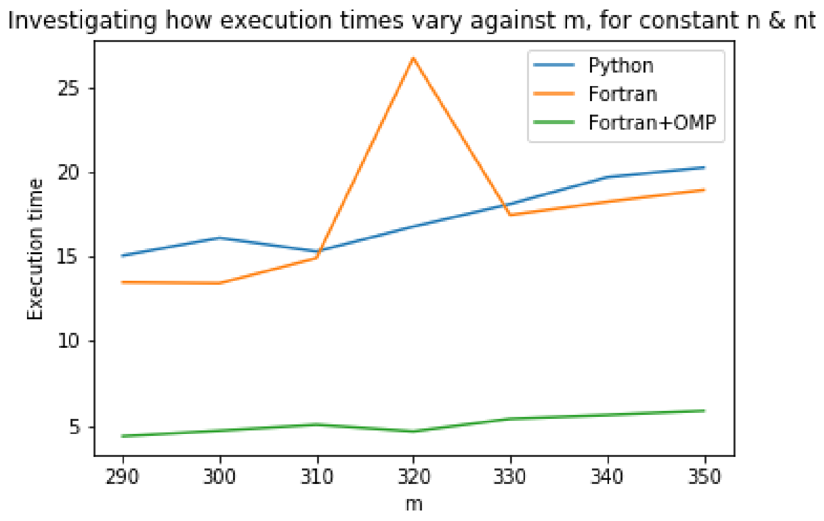

Figure 4: Analysing the effect of m on execution times in closer detail. Here $(m, nt) = (200, 300)$

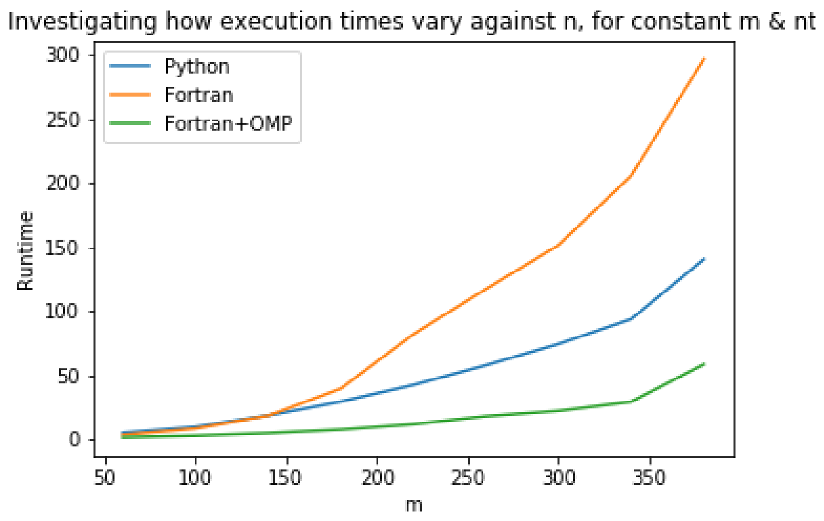

Figure 5: Analysing the effect of n on execution times. Here $(m, nt) = (200, 300)$

Question 3

We can see in figure 6 the measure of correlation as described in the project description, C(tau) against tau. The results generated from the Python and Fortran implementations are very similar, differing slightly due to the random numbers used earlier in the program. The results from the Fortran+OMP implementation is slightly greater on average, with a bigger variation than the other two.

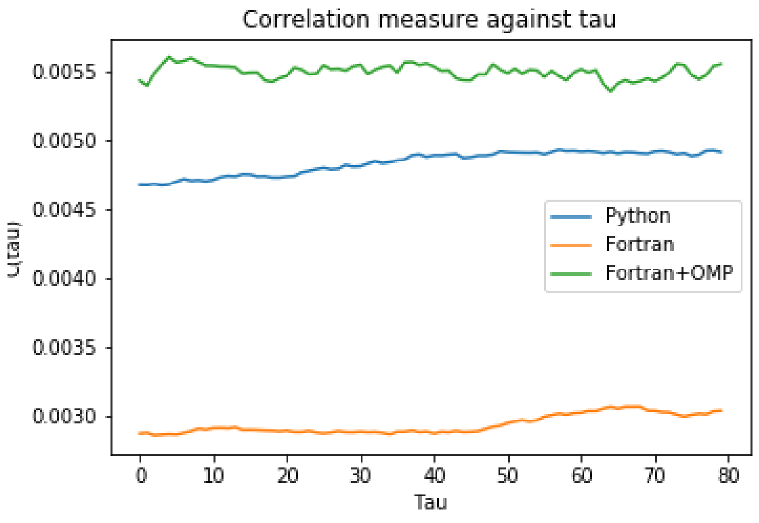

Figure 6: Relationship between tau and C(tau)
, C(tau) as defined in the project description.