Excercise 4

Abstract

This project consisted evaluating the performances of the algorithm developed for the matrix multiplication done in the previous weeks. This has been done through the use of a python3 script and the gnuplot software, plus introducing new subroutines to handle the input and output from file of the results.

1 Theory

Let T_A be the time needed for the execution of the algorithm $A(N)$, subsequently the dependence of N by A is translated on the execution time resulting in $T_A(N)$. Suppose:

$$T_A(N) = \sum_{i=0}^{i=\rho} a_i N^i \tag{1}$$

hence we get

$$T_A(N) = N^\rho (\sum_{i=0}^{\rho-1} a_i N^{i-\rho} + a_\rho) \tag{2}$$

$$\log(T_A(N)) = \rho \log(N) + \log(\sum_{j=-\rho}^{-1} a_{j+\rho} N^j + a_\rho) \tag{3}$$

and substituting the following

$$x' = \log(N) \qquad y' = log(T_A(x')) \qquad \epsilon(x) = \sum_{j=-\rho}^{-1} a_{(j+\rho)} e^{jx'} \tag{4}$$

we obtain

$$y' = \rho x' + \log(\epsilon(x') + a_\rho) \tag{5}$$

Noticing that

$$\lim_{x' \to +\infty} \epsilon(x') = 0$$

it is possible to fit [eq.5] linearly for "enough large" N.

2 Code Development

The free parameter N chosen for the algorithm represents the number of rows/-columns of two square matrices used as input for the matrix multiplication algorithms. To set N, I developed a function, **read_dim**, inserted in the **matrixqic** module which sets N equal to a parameter that is read from file. Moreover **read_dim** uses the **breakifn** subroutine, the debug function developed for the exercise 3, to verify the existence of the input file.

```
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 !read dimension from file.
 FUNCTION read_dim(path) RESULT(dim)
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 character(len = *) :: path
 integer :: dim
 logical :: file_exist = .FALSE.

 INQUIRE(FILE=path, EXIST=file_exist)
 call breakifn(path // "does not exists",file_exist, debug_var)
 open(unit = 3, file = path , status = "old" )
 read(3, fmt= "(i8)") dim
 close(3)

 END FUNCTION
```

Then N is fed to a function that generates random matrices, and finally these into the various subroutines coded to perform the matrix multiplication: **matmul1**, **matmul2**, **matmul3**, **matmul4**. Their implementation can be seen in the exercise 3 report.

The output of the code consists in the time took to compute the matrix multiplications using the different algorithms. The results are then appended in multiple .txt files, one for every matrix multiplication algorithm. These contains two columns: the first stores the parameter N and the second the computational time. The code implementing this functionality has been developed again in the **matrixqic** module using a subroutine called **write_float_out**.

The python3 script is necessary to compile, execute the fortran90 code and to set a convenient hierarchy of folders to store the results. This performs two

loops: the first varies the compiler optimization used to compile the fortran90 code, while the second one is necessary to change the dimension of the input matrices. The code follows:

```
import os
Ns = [50*i+1 for i in range(1,30)]
print(Ns)
optimizers=["", " -O1", " -O2", " -O3"] #vary optimizer
folders=["optn", "opt1", "opt2", "opt3"] #folder 4 optimizer
outfiles=["matmul.txt","matmul1.txt" #files gen by ./test
,"matmul2.txt","matmul3.txt","matmul4.txt"]
 #check if the output dir is present
if not os.path.exists("./out"):
    os.makedirs("./out") #if not make it
for i in range(4): #vary optimization of compiler
    #select optimizer and folder
    cdir=folders[i]
    copt=optimizers[i]
    #compile
    os.system("gfortran"+copt+" test3.f90 -o test")
    if True :
        for n in Ns:
            #change dim of matrices
            print ("dim = " , n , ":")
            f = open("dim.txt", "w+")
            f.write(str(n))
            f.close()
            #run program
            os.system("./test")
    #once collected the data fit and plot everything
    os.system("gnuplot ./g_script.txt")
    #check if the output dir is present
    if not os.path.exists("./out/"+cdir):
        os.makedirs("./out/"+cdir) #if not make it
    #move everything into the respective folder
    for f in outfiles:
        os.system("mv ./out/"+f+" ./out/"+cdir )
    os.system("mv ./file.png ./out/"+cdir )
    os.system("mv ./fit.log ./out/"+cdir )
#script generating report plot
os.system("gnuplot ./g_script2.txt")
```

Additionally a gnuplot script has been developed to configure the appearance of the resulting plots and fit the data. In particular the fit has been done using the following function.

$$T(N) = exp(a \log N + b) \tag{6}$$

The gnuplot script is called **g_script** and its invocation can be seen in the python3 code.

3 Results

The result showed that the built-in fortran function that performs the matrix multiplication has the lowest computational time with respect to the custom defined one. The main issue has been found in the **matmul4** results. This, if the code is compiled using optimization flags, at a certain value of N rises rapidly, introducing a sort of step in the points [Fig.1]. The reasons of this behaviour remain unknown. Overall the custom defined matrix multiplication routines respect the prescribed fit and show minimum differences in the result, although the time efficiency ranking remains unchanged varying the optimization. This suggests that even if the difference is small, a loop over the columns has a different computational cost than one on the columns.

4 Self Development

This project has been a test bench to prove my knowledge about gnuplot, and the use python as an interface to execute commands in the gnu terminal. Moreover since I performed tests on the algorithms using loops it would be interesting to rethink the python script to perform the same computations in a "multithreaded" fashion, allowing the computation to run in parallel to save time. However it is not clear if this procedure would change the results of the simulation. Additionally it should be a good idea to take in account the behaviour of $\epsilon(x')$, not to be more precise in the fit, but to be sure to have chosen proper values of N to perform the analysis.

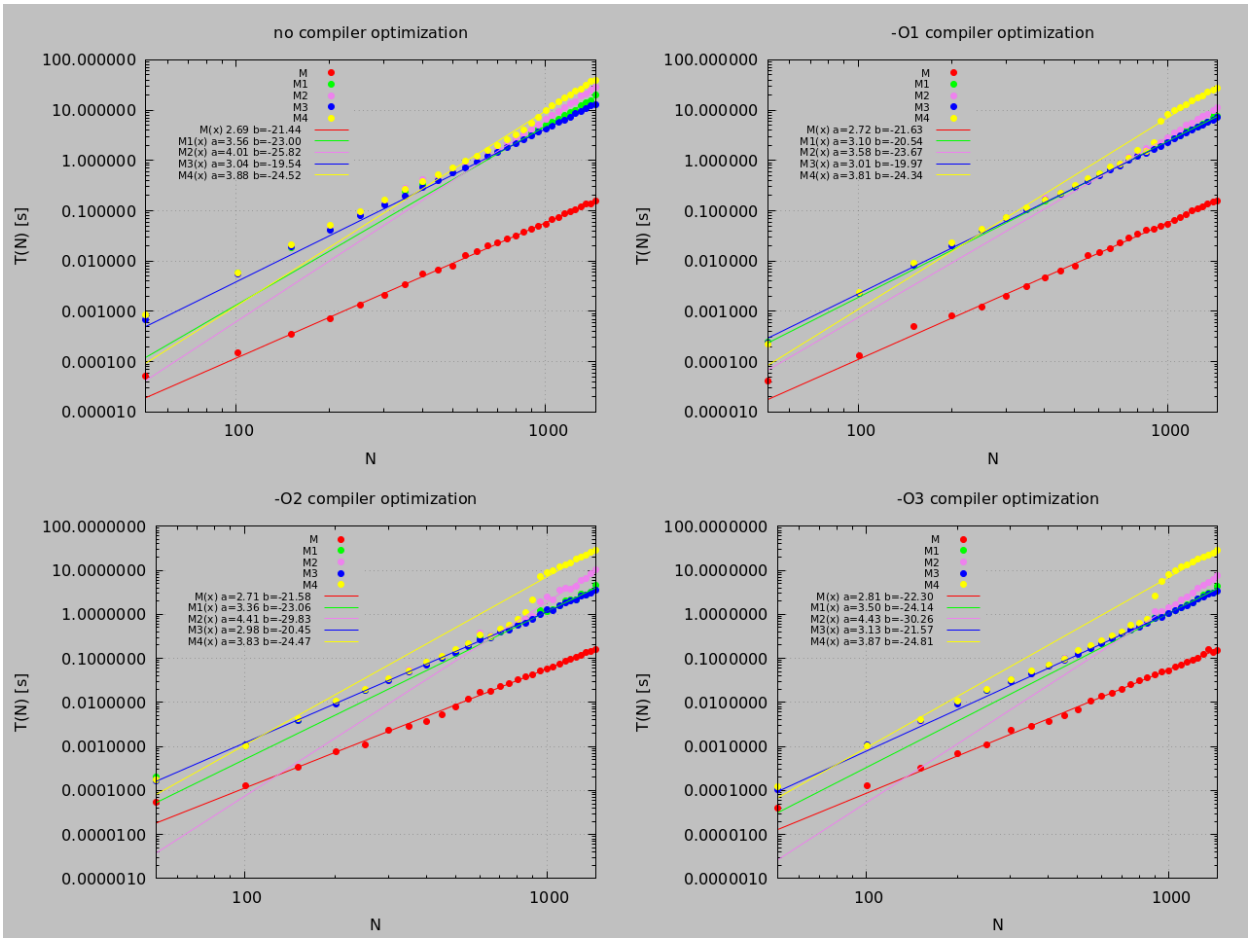

Figure 1: Figure showing the result of the fit for various optimization flags. In the y axis is showed the time and in the x-axis the number of rows/columns of the input matrices. Every plot is represented with a log-log scale. M represent the built in fortran routine, while M1, M2, M3, M4 represent **matmul1-2-3-4**. The lines represent the output functions for the fits named with the same names showed above. The parameters a and b are the ones found in 6. It is possible to appreciate also the strange behaviour of **matmul4** (yellow line/dots). This graph has been realized not with **g_script** but with a clone one, that exploit the multiplot functionality of gnuplot.