Exercise 4
Automated runs, plots and fits

Michele Guadagnini - ID 1230663

November 2, 2020

Abstract

This exercise is about using a *FORTRAN* compiled executable in a *Python* script to run it multiple times with different parameters. Then, the data produced are plotted and fitted automatically by using a *gnuplot* script.

1 Theory

The execution time of a certain program is the time spent by the CPU while completing the task. It depends on the specific hardware on which it is executed (processor speed, disk and memory speed). Also, it can make use of these resources with more or less efficiency. Optimizations to achieve better efficiencies can be applied on the source code and, for compiled languages, at compile time. *FORTRAN* in particular provides many different optimization levels; the ones used in this assignment are:

- "-O0": it applies no optimization at compile time (default);
- "-O2": it activate some optimizations and make the compiler take longer to finish;
- "-Ofast": it activates all the optimizations.

In this exercise the execution time of a matrix multiplication is measured for three different methods and by scaling up the size of the matrices (for simplicity in this case matrices are assumed to be square). A good model to fit the curves obtained from these data can be a power law model, that can be transformed into a linear fit by taking the logarithm of both x and y values.

2 Code Development

2.1 Design and Implementation

The implementation started from modifying the *FORTRAN* code from the previous assignment by creating the function that reads the desired matrix size from a file, implementing it in the *LoopMultiply* module. It also checks the size to be a positive integer number, otherwise it prints an error message.

Matrices are set to be square with numbers between 0 and 1 of *DOUBLE PRECISION* type. The name of the file containing the matrix size is passed to the program as a command argument together with the output file names for the three different methods and the optional debug flag.

Then the subroutine to save the results has been implemented (*WriteExecTimeToFile*). It receives as input the *unit* number of the output file, the matrix size and the execution time measured and it writes them in the file side by side.

Also a subroutine called *CatchError* has been added and used in the program alongside the *CheckPoint* one in the module *Debugger*. This new subroutine prints a message when an error occurs and, if the error is set to be a *fatal* error through the dedicated logical variable, it prints the same message also in a file in order to expose the error to an eventual external script and stops the execution.

Once finished the implementation of the program performing the multiplication, a script to call it repeatedly with different arguments has been created using Python. The script is called *Ex4-Guadagnini-PyScript-CODE.py* and its main part is reported in the following listing:

```python
Nmin = 50
Nmax = 2001
Incr = 50

## begin ##
start_time = time.time()
print("Matrix-matrix multiplication test")

# resetting the errorfile
open(errorfile,"w").close()
# ensure existence of the data folder
if not os.path.isdir(folderpath):
    os.makedirs(folderpath)
# resetting the output files:
print("  resetting the results files...", end='')
for jj in range(3):
    for ii in range(3):
        open(folderpath+"/"+methods[ii]+flags[jj]+".txt", "w").close()
print(" Done.")

for N in range(Nmin,Nmax,Incr):
    # overwriting the size file
    print("\nMatrix size set to "+str(N))
    with open(sizefile, "w") as file1:
        file1.write(str(N))

    # Matrix-matrix multiplications and data storing
    for jj in range(3):
        call([ "./MatrixTest"+flags[jj]+".x", sizefile,
               folderpath+"/"+methods[0]+flags[jj]+".txt",
               folderpath+"/"+methods[1]+flags[jj]+".txt",
               folderpath+"/"+methods[2]+flags[jj]+".txt", debug ])
        #check for errors
        if (os.stat(errorfile).st_size != 0):
            sys.exit("An error occured during the fortran program execution
, causing it to stop. Please check "+errorfile+" for details. Exiting
...")
        else:
            print("  multiplication with -"+flags[jj]+" flag successful")

print("Time elapsed from start [s]: %s " % (time.time() - start_time))
## end ##
```

Listing 1: Multi-Run python script.

It starts by resetting all the output files, then it writes the matrix size in *MatrixSize.txt* starting from 50 until 2000 with steps of 50. Since the program has been compiled with three different optimization flags (as explained above), the script runs all the three executable using a *for* loop passing the respective output files. If the multiplication program encounters an error, the python script is aware of this by looking at the size of the error file and prints a message while stopping the execution.

The last part of the exercise was to use *gnuplot* to plot the obtained curves and fit them automatically through a python script. The set of instruction used for the fit are in the file *AutoFit.gnu* that is passed as argument when running *gnuplot* from the python script *Ex4-Guadagnini-AutoFit.py*. The script simply lists all the ".txt" files in the current folder and pass them one by one to *gnuplot* that execute the fit, and changes the name of the *fit.log* file. To perform a linear fit, gnuplot takes the logarithm of both the matrix size and the timings and applies the linear model:

$$F(x) = ax + b$$

The following listing reports the gnuplot instruction set for the fits:

```
datafile = ARG1
length = strlen(datafile)
method = substr(datafile, 0, length-4)
ff(x) = a*x+b
set key right bottom
set yrange [:2.2]
set xlabel "Log_{10}(N)"; set ylabel "Log_{10}(Time)"
#set title sprintf("%s%s", "Plot and Fit: ",datafile)
set grid
fit ff(x) datafile u (log10($1)):(log10($2)) via a,b
set label 1 "model: f(x) = a*x+b" at 1.7,1.5 font "arialbd,18"
set label 2 sprintf("  a = %3.4f", a) at 1.7,1.1 font "arialbd,18"
set label 3 sprintf("  b = %3.4f", b) at 1.7,0.7 font "arialbd,18"
set term pdf color enhanced size 4,4
set output sprintf("%s%s%s","Fit_",method,".pdf")
plot datafile u (log10($1)):(log10($2)) w p lc "red" pt 1 ps 1 title method
    , ff(x) w l lc "blue" lw 1.2 title sprintf("%s%s", method, " fit")
```

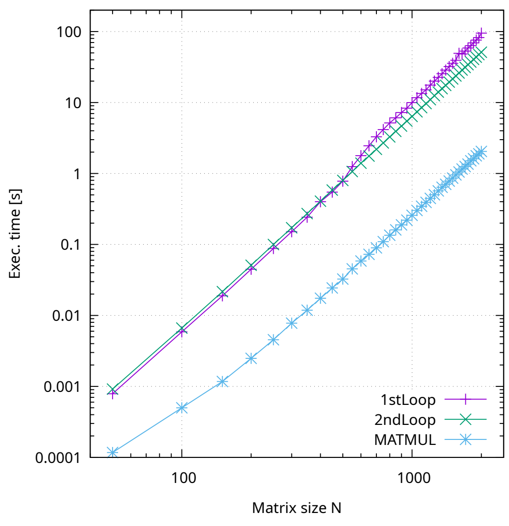
Listing 2: gnuplot script performing a fit.

2.2 Debug and Test

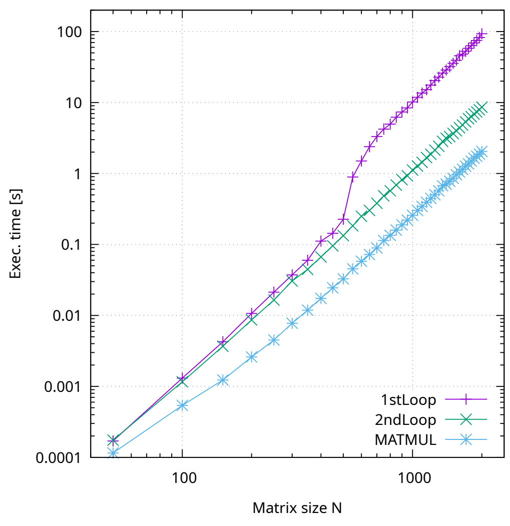
The multiplication program has been tested before the use in the script by compiling and calling it in the terminal with the command:

```
gfortran *CODE.f90 -o MatrixTest.x
./MatrixTest.x MatrixSize.txt 1stLoop.txt 2ndLoop.txt MATMUL.txt
```

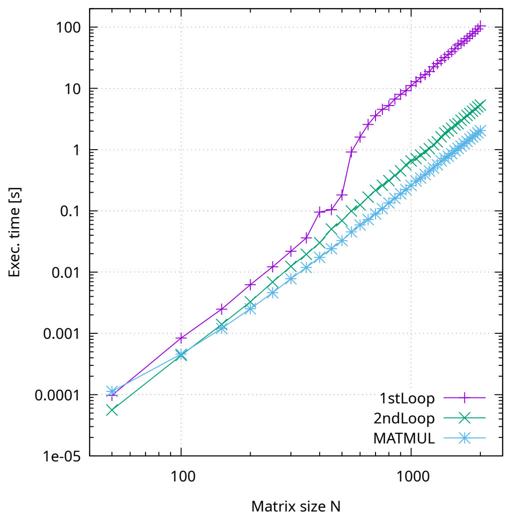
To test the error handling capabilities of the code, the python script has been temporary modified to write as matrix size a negative float number and also to not create the file at all. As expected, the FORTRAN executable encountered an error since it was not able to properly read the matrix size or the file containing it; through the function *CatchError* a message was printed on screen and in this case also in the dedicated file *FATAL_ERRORS.txt*. By checking that this file was no more empty, also the python script stopped and printed the coded message, suggesting to look on that file for details on the error.



(a) All three methods with -O0 flag



(b) All three methods with -O2 flag



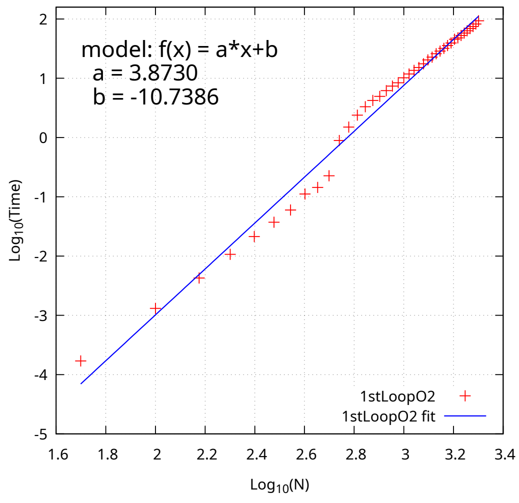
(c) All three methods with -Ofast flag

Figure 1: Time scaling for different methods and optimizations

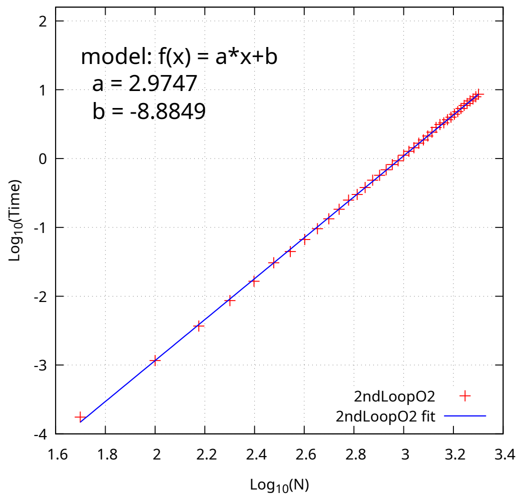
3 Results

In Figures 1 are presented the pictures of the curves obtained for different methods and optimizations. The scale on the y axis has been set to "log" since values space between 7 orders of magnitude. Comparing the figures it can be seen that the method that mostly benefits from optimization flags is *2ndLoop*, while *1stLoop* only have a small improvement until size 500 and *MATMUL* seems to be not sensitive to optimization flags.

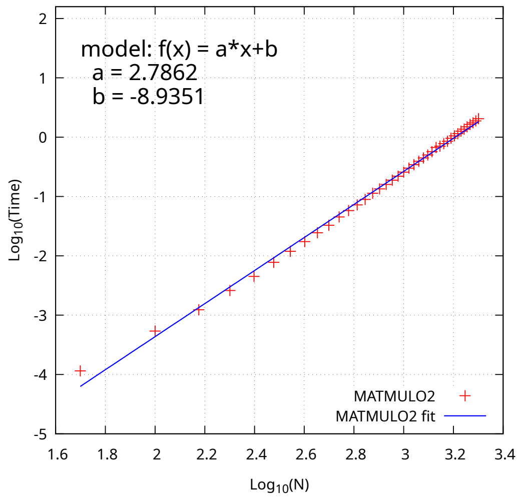
Figure 2 reports the fits of the timings for the three methods compiled with *-O2* flag. In particular Figure 2a shows a bad fit due to the jump in execution time for *1stLoop* method around matrix size of 500 ($Log_{10}(500) \approx 2.7$). The other two fits (Figures 2b and 2c) seems to be more accurate.



(a) Fit of *1stLoop* data with -O2 flag



(b) Fit of *2ndLoop* data with -O2 flag



(c) Fit of *MATMUL* data with -O2 flag

Figure 2: Fitted curves for the three methods with -O2 flag.

4 Self-evaluation

Things learned while completing this assignment are:

- how to call a compiled program from a python script by using the libraries *subprocess* or *os.system*;
- how to use *gnuplot* interactively and also in a script.

It would have been better to run the program with a non-equally spaced sequence of values for the matrix size to have better linear fits and plots in double log scale. Also, it could be useful also to automatize the plot of the residuals of the fits in order to check their quality and reliability.