Multi-run script and Automated fits

Davide Maniscalco

November 5, 2019

Abstract

In this exercise the previous fortran code, exercise 3 of week 1 about matrix-matrix multiplication, is rewritten in order to can read from file the matrix dimension and to write to file the CPU-time results. Then, results are shown and some fits are made with gnuplot. Finally, a python script is written in order to write in a once all the matrix dimensions to be given as input to the fortran code and to automatize all the previous processes.

NOTE ABOUT THE COMPILATION

- The delivered fortran code `'Ex4-Maniscalco-CODE.f90'` uses the external `CHECKPOINT MODULE`: see the note on the previous paper for details about the compilation.
- The executable file of the fortran program must be called `'Ex4-Maniscalco.out'` if it is wanted that the python script is able to launch it automatically.
- The name of the delivered gnuplot script `Ex4-Maniscalco-gnuplot` must not to be changed if it is wanted that the python script is able to launch it automatically.
- The python script is written in Python 3 and it must be run the following way: `python3 Ex4-Maniscalco-SCRIPT.py`

1 Code development

1.1 Fortran code

The previous code `Ex3-Maniscalco-CODE.f90` is recovered and modified the minimum necessary. First of all, a do-loop is implemented with the purpose of reading from file named 'inputdim.dat' a list of arbitrary length of matrices' dimensions (only squared matrix products were made in this specific code). Its beginning is:

```
OPEN(UNIT=77,FILE='inputdim.dat',status='old')

ss = 1                               !needed at bottom to open output file

DO
READ(77,*,iostat=reason) nndim
  IF(reason<0) THEN                  !reason<0 means that it reached the end
    of the file
  EXIT
  ELSE
```

If the `ELSE` statement is reached the program runs and makes all the multiplications for the given input. The do loop works until the last line of the input files is reached, fact that is reported by the `reason` variable. The input matrix dimension is valid as the number of rows and the number of columns of all input and output matrices. Then the program performs the matrix-matrix product by row, by column, and with matmul, and calculates the time spent for the processing. Then the old code is modified with the purpose of writing in three different files the times:

```
!-------------------------------------------------------
! Times printing
! ------------------------------------------------------

! first iteration: if the file already exists, it is replaced. Other
   iterations: append
IF (ss==1) THEN
OPEN(UNIT=10,FILE='byrow.dat',status='replace',access='append')
OPEN(UNIT=20,FILE='bycol.dat',status='replace',access='append')
OPEN(UNIT=30,FILE='matmul.dat',status='replace',access='append')
ELSE
OPEN(UNIT=10,FILE='byrow.dat',status='old',access='append')
OPEN(UNIT=20,FILE='bycol.dat',status='old',access='append')
OPEN(UNIT=30,FILE='matmul.dat',status='old',access='append')
END IF

WRITE(10,*) T2-T1
WRITE(20,*) T4-T3
WRITE(30,*) T6-T5

 CLOSE(10)
 CLOSE(20)
 CLOSE(30)
```

The files 'byrow.dat','bycol.dat','matmul.dat' containing the times of the multiplications by row, by column and with matmul respectively are created. The variable `ss` has value `1` only at the first iteration: here, if there exist in the folder some files with the same name of the ones that have to be created, these are replaced. In the next iterations, instead, the program writes in the new files created at the first iteration. Finally, at the end of the program the current matrix dimension is printed to terminal, in order to allow to know about the progress of the program while running.

1.2 Python script

The python script can perform the following tasks:

Generate a file named named 'inputdim.dat'. This contains in a column all the integer numbers between two extremes (the latter excluded) with a given step; these can be changed in the script. The list is written with the command `N=np.arange(Nmin,Nmax,step)` while the `.dat` file is created with the `np.savetxt('inputdim.dat',N,fmt='%u')` command.

Run the program 'Ex4-Maniscalco.out' This is meant to be the one described in 1.1. This is done with the command `subprocess.call('./Ex4-Maniscalco.out')`

Run the gnuplot script 'Ex4-Maniscalco-gnuplot' This is the script created with the purpose of making graphs and fits, it will be described in 1.3. This is done with the command `os.system('gnuplot Ex4-Maniscalco-gnuplot')`

The program asks as first to the user for the option that he needs. This way one can only fit already existing data or only generate the input file from the python script.

1.3 Gnuplot script and fits

The first purpose of the gnuplot script is of plotting the cpu-times of the three multiplication methods as a function of the matrix dimension. All these results are contained in different files, but must be put together. Therefore it is decided to create three virtual files with the `paste` command, e.g.: `Rows = "<paste inputdim.dat byrow.dat"`. After setting linetype, labels, title, key position and colors, the plot is made with: `plot Rows title 'By row', Cols title 'By col',Matmul title 'Matmul'`.
The second purpose of the script is of performing fits of the time of the three methods as a function of the matrix dimension. Since the dependence is a power function passing through the origin, three linear functions are defined, and then a logarithmic fit is made.

$$y = mx + q \qquad \log(y) = m\log(x) + q \qquad y = x^m e^q$$

This way the coefficient m becomes the power of the variable and e^q the coefficient. One example of the commands for the fit and for the plot is the following:

```
Rows = "<paste inputdim.dat byrow.dat"
f(x) = a*x+b
fit f(x) Rows u (log($1)):(log($2)) via a,b
plot Rows u (log($1)):(log($2)) title 'By row',f(x) title 'By row fit'
```

The operations on the file columns are made with the special command $.
The two graphs are saved as two different `.png` images, e.g.: `set output 'cputime_matrixmatrix.png'`.

2 Results

A first run of the program with the input matrix dimensions $N_{\text{start}} = 50, N_{\text{stop}} = 2501, \text{step} = 50$ is made, producing the results shown in Figg. 1,2 and in Tab.1.

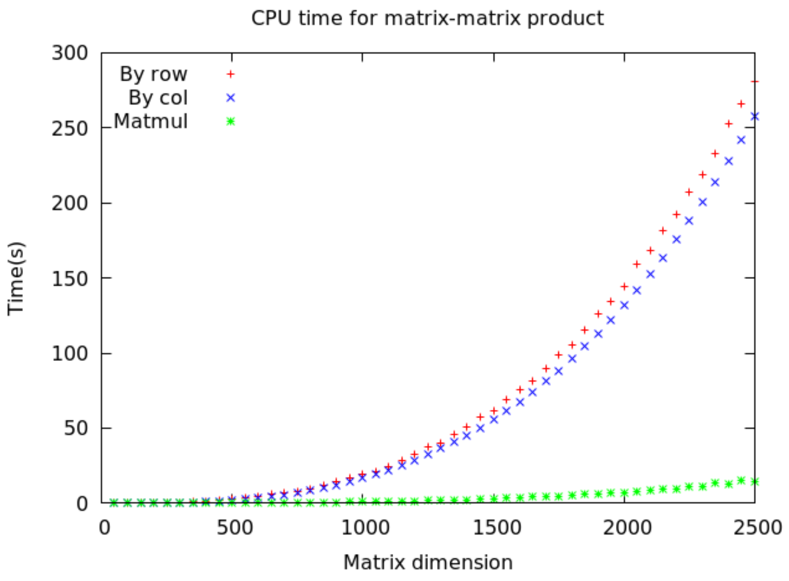

Figure 1: CPU times vs Matrix dimensions in the range [100,2500]

As we can see, the predicted exponent of the power law is around 3 for all the methods, that is what is expected from the theory. What is much different between the two methods is the coefficient of the power, that makes matmul $\simeq$ 170 times quicker than the other

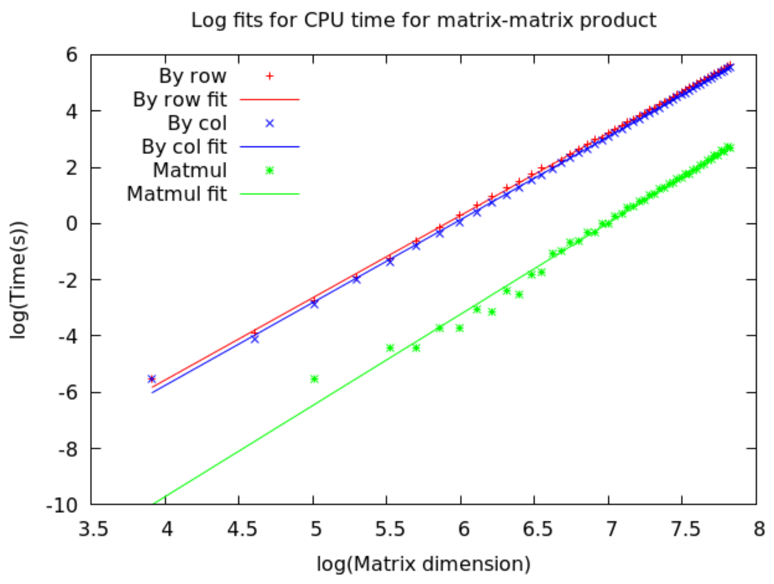

Figure 2: Logarithmic fits for CPU times vs Matrix dimensions in the range [100,2500]

	m	$m_{\text{error}}(\%)$	q	$q_{\text{error}}(\%)$
By row	2.93	0.36	-17.28	0.07
By column	2.94	0.45	-17.56	0.09
Matmul	3.24	1.50	-22.7	0.24

Table 1: Fit results for the points in the range [100,2500]

methods.
Now it is wanted to do again these fits but considering the biggest possible difference between N_{min} and N_{max}. A very rough estimation about the maximum dimension can be done. Each matrix contains N^2 elements 8 bytes each, and every time 5 matrices have to be allocated (two as input and three for the products). In the hypothesis of occupying 4GB of the RAM (a PC with a RAM of 8GB is available by the author), this leads to a maximum dimension of:

$$N = \sqrt{\frac{4 \cdot 10^9}{5 \cdot 8}} = 10000 \,.$$

It is needed now to calculate the time needed by the program to perform all the calculations. As explained in section 1.3 the relation between the time and the matrix dimension follows a power law. The total time is therefore calculated summing the time for all the methods for all the dimensions, using the coefficient of Tab.1.

$$y_{\text{tot}} = \sum_x x^{2.93} e^{-17.28} + \sum_x x^{2.94} e^{-17.56} + \sum_x x^{3.24} e^{-22.7}$$

However with the options ($N_{\text{start}} = 2600, N_{\text{end}} = 10000, step = 500$) the total time was about $\simeq$ 43h, too long for a week-long assignment. Therefore, it was decided to run the program with the options ($N_{\text{start}} = 2600, N_{\text{end}} = 4400, step = 100$), that took about seven hours and half. Then the result were put together with the ones about the previous range [100,2500]; the final graphs and fits are shown in Figg.3,4, Tab.2.

As before, the power coefficient for all the methods is about 3, while the whole coefficient before the power is 170 times smaller for the intrinsic function matmul.

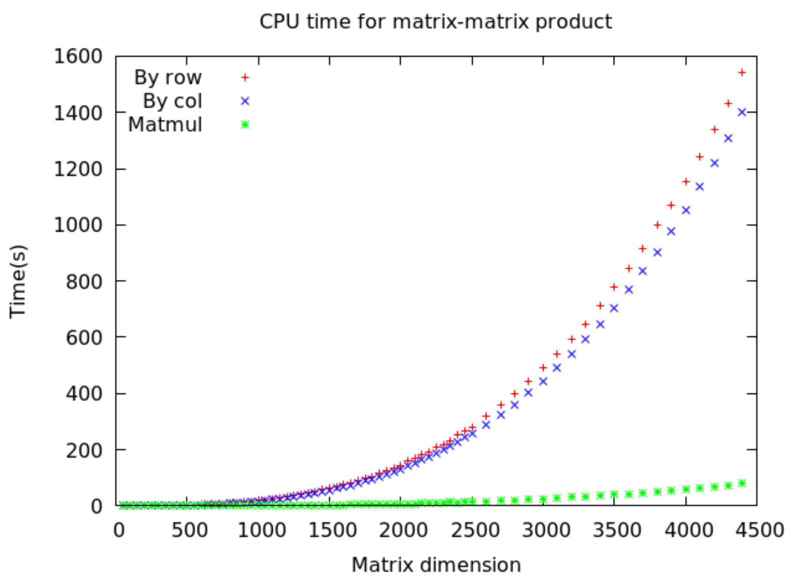

Figure 3: CPU times vs Matrix dimensions in the range [100,4400]

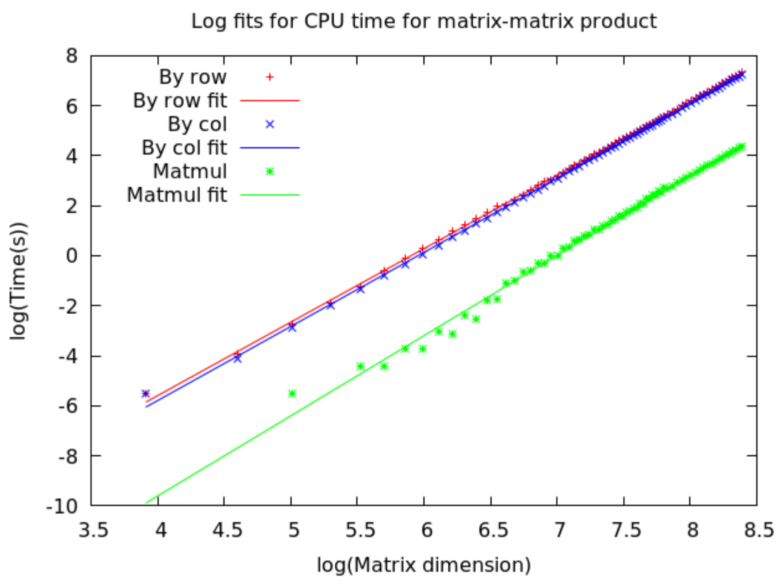

Figure 4: Logarithmic fits for CPU times vs Matrix dimensions in the range [100,4400]

	m	$m_{\text{error}}(\%)$	q	$q_{\text{error}}(\%)$
By row	2.940	0.25	-17.35	0.30
By column	2.963	0.31	-17.64	0.07
Matmul	3.21	0.96	-22.7	1.02

Table 2: Fit results for the points in the range [100,4400]

3 Self evaluation

All the tasks have been achieved. Maybe it would have been possible to reach a bigger maximum matrix dimension taking less steps.