Assignment 1

Quantum Information and Computing Course 2022/2023

Massimo Colombo

Department of Physics and Astronomy "Galileo Galilei"
Degree in Physics

October 30, 2022

Exercise 1

First exercise's request:

- Working directory: Quantum Information Exercises
- Code editor: Visual Studio Code
- Test job: Hello World

```
Quantum Information Exercises > Ex_1 > Fortran code > Exercise1.f90
program hello
    print *, 'Hello World!'

end program hello
```

Exercise 1: Code

It works!

```
Air-di-Massimo:Fortran code massimocolombo$ gfortran Exercise1.f90 -o Exercise1
Air-di-Massimo:Fortran code massimocolombo$ ./Exercise1
 Hello World!
Air-di-Massimo:Fortran code massimocolombo$
```

Exercise 1: Results

Exercise 2

To understand finite precision in FORTRAN, we were requested to do the following calculations, using different Data-types:

- with INTEGER*2 and INTEGER*4: $2000000 + 1$
- with REAL*4 and REAL*8: $\pi \cdot 10^{32} + \sqrt{2} \cdot 10^{21}$

In the first case, we expect an overflow: "2000000" does not belong to the definition range of an integer in two bytes $[-2^{15}, 2^{15} - 1]$.
In the second case, the significant figures depend on the precision used.

RESULTS:

```
Using Integer*2:
The sum of     2000000 and           1 is -31615
Using Integer*4:
The sum of     2000000 and           1 is     2000001
Using Real*4:
The sum of   3.14159259E+32 and   1.41421360E+21 is   3.14159259E+32
Using Real*8:
The sum of   3.1415926535897933E+032 and   1.4142135623730950E+021 is   3.1415926536039354E+032
```

Exercise 2: Results

WARNING!

```
Error: Arithmetic overflow converting INTEGER(4) to INTEGER(2) at (1). This check can be disabled with the option '-fno-range-check'
```

0em

Exercise 3

The goal is to compare the execution times of 3 different algorithms that perform matrix product:

- 3for-loops corresponding to the "handmade/usual" matrix product (a).
- Same 3for-loops with inverted indices (b).
- The FORTRAN intrinsic function: "matmul" from Blas.

The execution time was measured thanks to the **cpu_time()** function.

```
CALL cpu_time(start)
        DO ii = 1, n_rows_AA
            DO jj = 1, n_columns_BB
                DO kk = 1, n_columns_AA
                    CC_1(ii,jj) = CC_1(ii,jj) + AA(ii, kk) * BB(kk, jj)
                ENDDO
            ENDDO
        ENDDO
    CALL cpu_time(finish)
```

(a) Usual matrix product.

```
CALL cpu_time(start)
        DO kk = 1, n_columns_AA
            DO ii = 1, n_rows_AA
                DO jj = 1, n_columns_BB
                    CC_2(ii,jj) = CC_2(ii,jj) + AA(ii, kk) * BB(kk, jj)
                ENDDO
            ENDDO
        ENDDO
    CALL cpu_time(finish)
```

(b) Inverted-Indices matrix product.

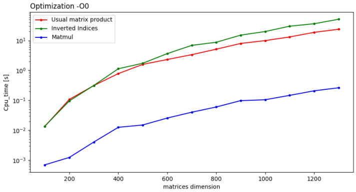
The subroutine **comparing_matrix(** A, B, ..., ϵ **)** has been defined to check the algorithm's correctness. Its purpose is to check that: $|A_{ij} - B_{ij}| < \epsilon \; \forall i, j$.

0em

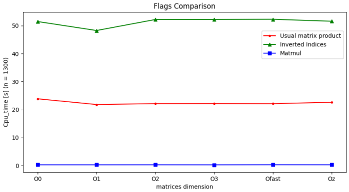
CPU Time and Optimization's Flags

Thanks to a Python script, it was possible to plot the execution time of the three algorithms as the size of the matrices varies (Fig. (a)).

Moreover, the comparison between the optimization's flag and the execution time of the algorithms (acting on $n \times n$-matrices) is shown in Fig. (b).



(a) Cpu time vs matrix dimension.



(b) Cpu time vs Flags

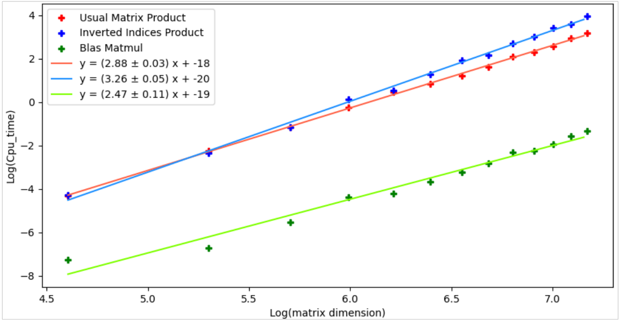
0em

Complexity

To conclude, the time complexity of the three algorithms is known:

- 3-for loops scale as $O(n^3)$.
- Matmul function from Blas scales as $O(n^{2.376})$.

An appropriate fit was carried out to check this behavior:



Time Complexity

0em