Quantum Information and Computing 2023-2024

Assignment 1

Maryam Feizi
2091504

November 2023

Exercise 1

home > maryam > projects > QIC > hello.f90

```fortran
! My first program in Fortran.
! This simple program is used to check if the Fortran code can compile.
! Author: Maryam Feizi

PROGRAM hello
      print *, 'Hello World'
END PROGRAM hello

```

```
maryam@topolbamaze:~/projects/QIC$ gfortran hello.f90 -o hello
maryam@topolbamaze:~/projects/QIC$ ./hello
 Hello World
maryam@topolbamaze:~/projects/QIC$
```

0 Sourcery

Exercise 1

home > maryam > projects > QIC > variables.f90

```fortran


program variables
    implicit none

    integer*2 :: int1, int2
    integer*4 :: int3, int4
    real*4 :: pi1 , real1 , real2
    real*8 :: pi2 , real3, real4

    int1 = 2000000
    int2 = 1
    print *, "With INTEGER*2:", int1+int2
    ! we want to sum in two different ways
    int3 = 2000000
    int4 = 1
    print *, "With INTEGER*4:", int3+int4

    pi1 = acos(-1.)
    real1 = pi1*10e32
    real2 = sqrt(2.)*10e21
    print *, "With single precision:", real1+real2

    pi2 = 4.D0*datan(1.D0)
    real3 = pi2*10e32
    real4 = sqrt(2.)*10e21
    print *, "With double precision:", real3+real4

    stop

end program variables
```

An overflow is occurring since **integer*2** variables is not in the range of two bytes which is : $[-2^{15}, 2^{15}-1]$

But there is not any error when we want to compute for **integer*4**

```
maryam@topolbamaze:~/projects/QIC$ gfortran variables.f90 -o variables
variables.f90:11:11:

   11 |     int1 = 2000000
      |           1
Error: Arithmetic overflow converting INTEGER(4) to INTEGER(2) at (1). This check can be disabled with the option '-fno-range-check'
maryam@topolbamaze:~/projects/QIC$ gfortran -fno-range-check variables.f90 -o variables
maryam@topolbamaze:~/projects/QIC$ ./variables
 With INTEGER*2: -31615
 With INTEGER*4:      2000001
 With single precision:   3.14159286E+33
 With double precision:   3.1415926363117528E+033
```

Exercise 3

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

- 3 for loops corresponding to the handmade/usual matrix product
- Same 3 for loops with inverted indices

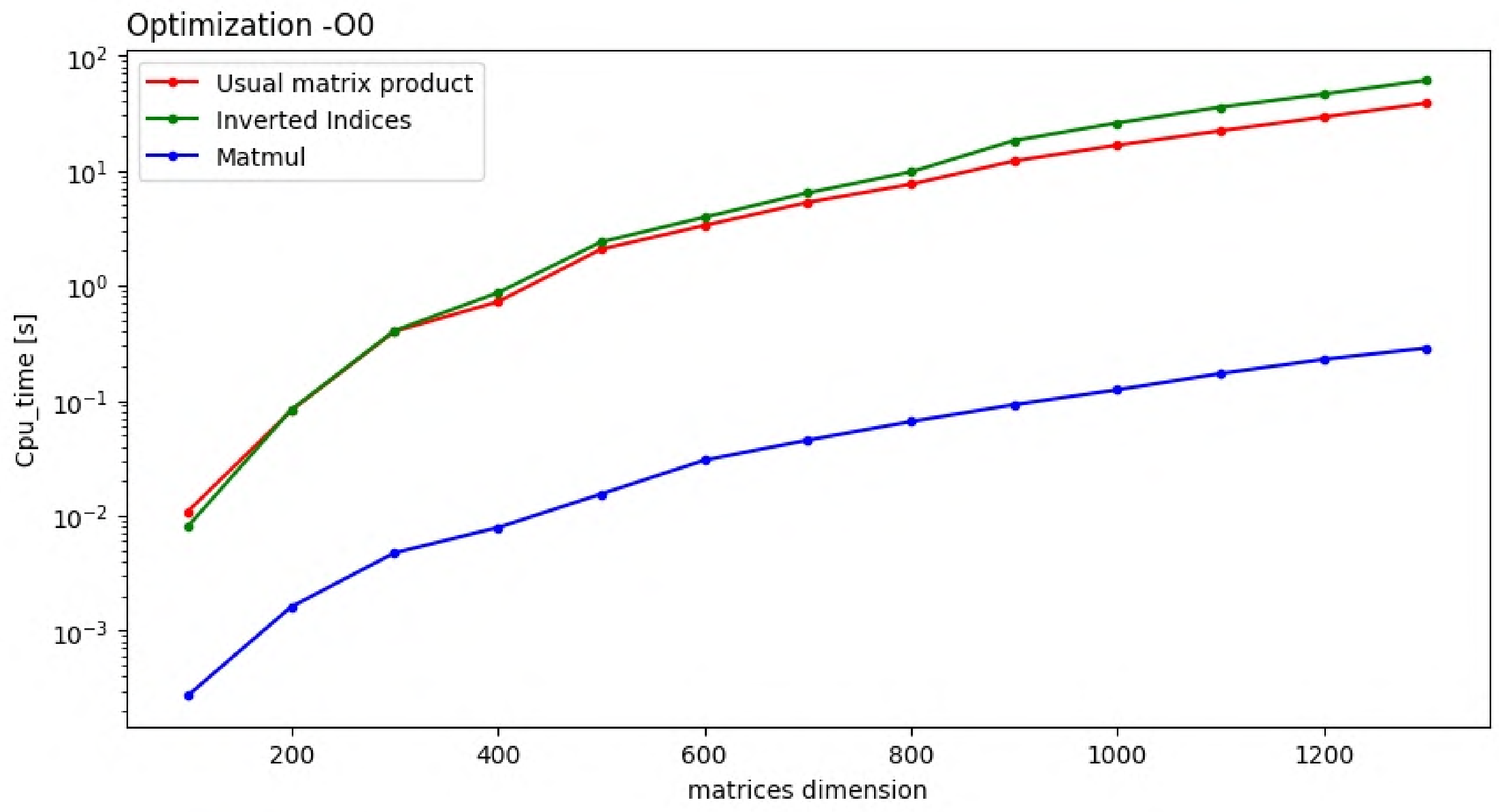
In this way the execution time was measured with **cpu_time()** function

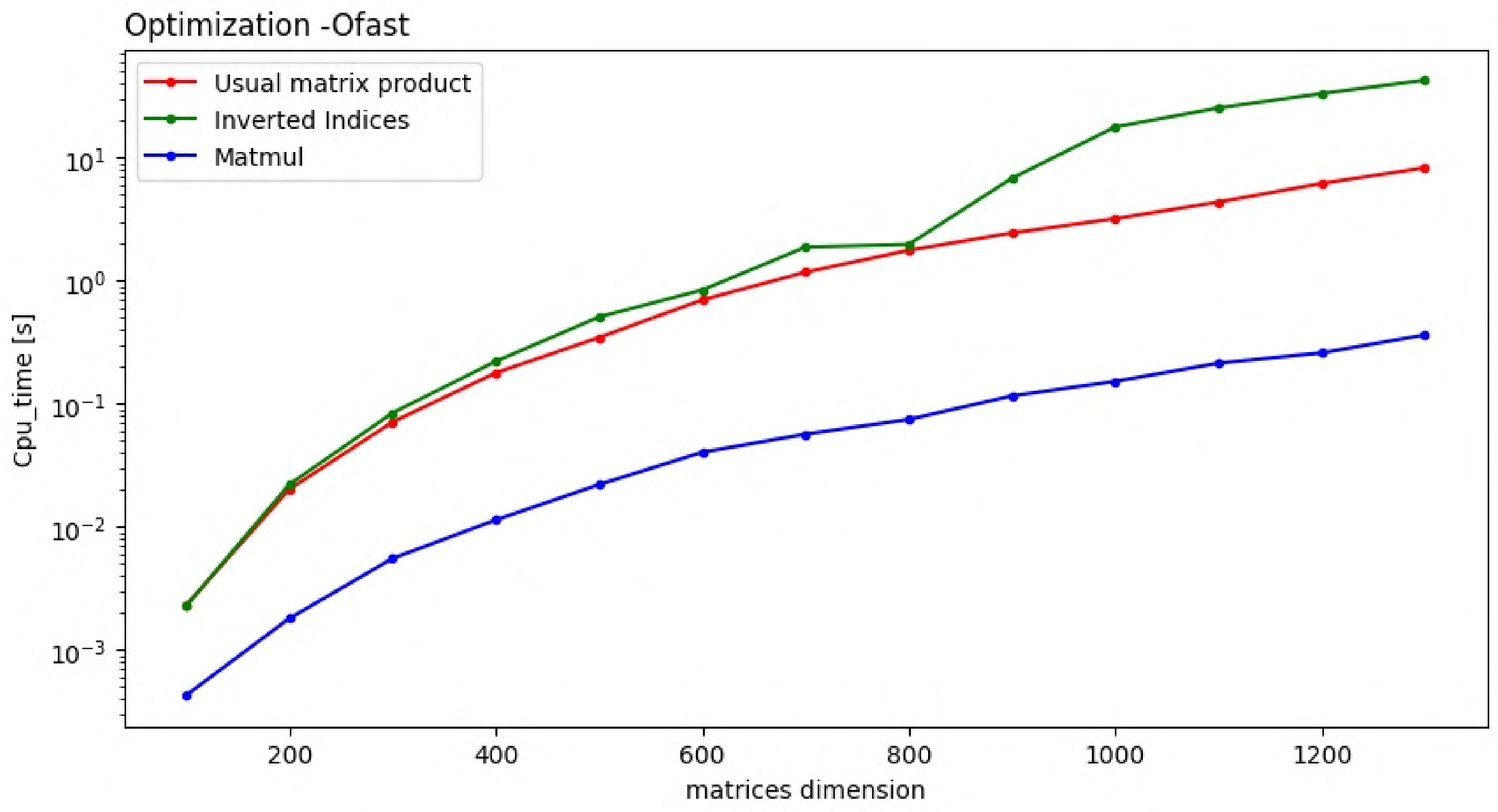
```
!1) 3 for-loops, Usual Matrix Product:

    ALLOCATE(CC_1(n_rows_AA, n_columns_BB))
    CC_1 = 0.0

    PRINT *, 'the matrix product through 3-for-loops (Usual way) is A*B ='

    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)
```

```
!2) 3 for-loops with inverted indices:

    ALLOCATE(CC_2(n_rows_AA, n_columns_BB))
    CC_2 = 0.0

    PRINT *, 'the matrix product through 3-for-loops with inverted indices is A*B ='

    !3 for-loops algorithm with inverted indeces
    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)
```

Exercise 3

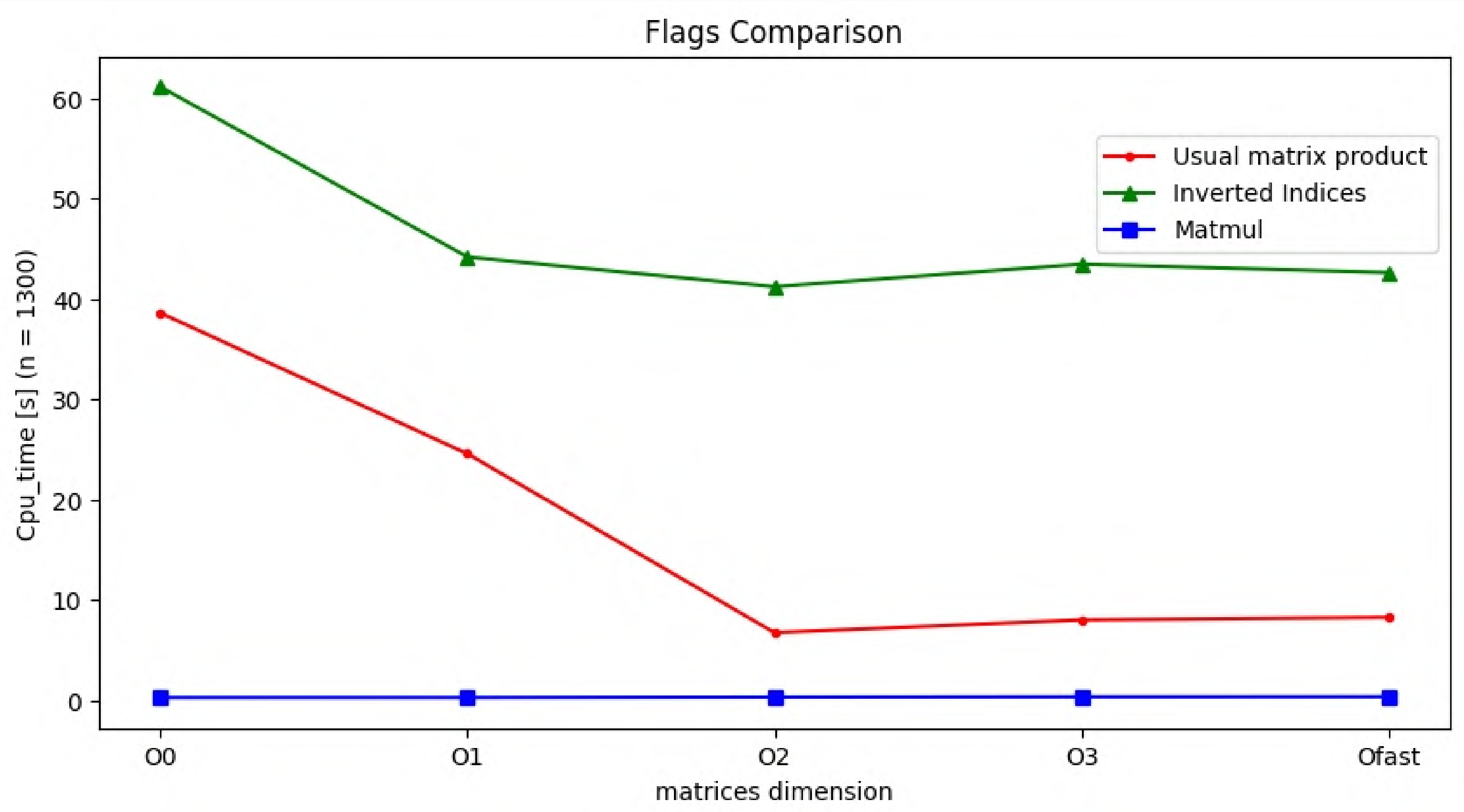
Thanks to the Python scripts , we can plot the execution time of the three algorithms as the size of the matrices .Also , plot cpu_time for different flags.



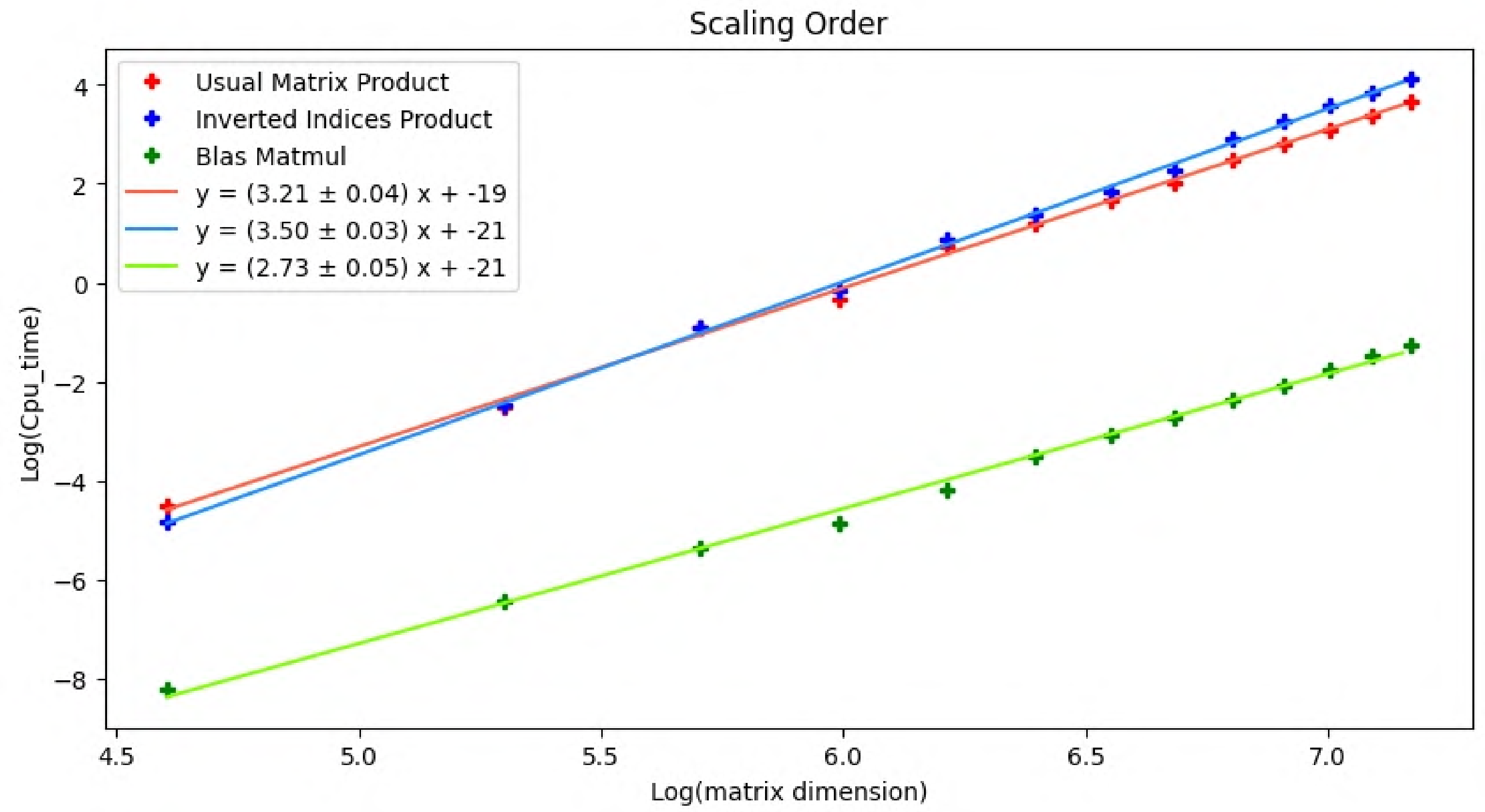


Exercise 3

Comparison between the optimization's flag and the execution time of the algorithm



Then compare time complexity of the three algorithms



Pitch

Want to make a presentation like this one?

Start with a fully customizable template, create a beautiful deck in minutes, then easily share it with anyone.

Create a presentation (It's free)