# Assignment 1

Amin Alimoradi

October 16, 2022

## 1 Introduction

This report illustrates a model that defines a group of experiments on matrix-matrix multiplication with the Fortran programming language. In this report, we will check the completion time of the process by Ifort, and Gfortran with and without OpenMP, optimization levels, and Basic Linear Algebra Subprograms (BLAS). Also, we will be testing the codes with Gfortran and Ifortran compiler to know how much they use CPU and Memory for calculating matrix-matrix multiplication. Methods will be tested when the computer is not swapping. The hardware that we run our tests on is :

**CPU Information**
Model name: : Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
CPU core is 4
Cache size is 6144 kb
CPU Max MHZ = 3500
CPU Min MHZ = 800
Threads per core is 1
**RAM Information**
Memtotal is 16263280 kb
Cached is 5104028 kb
Memfree 379620 kb
Memavailable 5475988 kb

## 2 Methods 1

First of all, we write our code by explaining step by step. Math.f90 is our main program for creating matrix-matrix multiplication,then we need to add a fillMatrix.f90 program. we perform compile the code with the Gfortran compiler by this the command: ( Gfortran -o amin.x math.f90 fillMatrix.f90 ) then in next line we run the project by this command: ( ./amin.x ). The process takes approximately 10 seconds by Gfortran compiler. On the other hand, we test this project with the Ifort compiler. Following that, we use this command: ( Ifort -o amin.x math.f90 fillMatrix.f90 ) then in next line we run the project : ( ./amin.x ).In this case, calculating takes around 0.51 seconds which is so faster than the Gfortran.

Second of all, we want to optimize the codes. First, we use a command which includes -On : (Gfortran -On -o math.f90 fillMatrix.f90 ),then we define an auxilary varibale as n which the range of that is between 0 and 3 (including both 0 and 3). Also, we use all commands for Ifortran too. In this report we check four optimization levels as follows: **-O0** is No optimization (the default); generates unoptimized code. **-O1** optimizes reasonably well but does not degrade completion time significantly in Gfortran but it takes long time for Ifortran.**-O2** is full optimization and it was a few seconds less than default but in Ifortran it was almost same.**-O3** is full optimization and so fast in both compilers than default.

The results are shown in the following table:

| Compiler | No Optimization | -O0 | -O1 | -O2 | -O3 |
|---|---|---|---|---|---|
| Gfortran | 10.604 | 10.807 | 8.668 | 8.932 | 4.164 |
| Ifortran | 0.475 | 10.474 | 8.085 | 0.475 | 0.164 |

| Hardware | without OMP | -O0 | -O1 | -O2 | -O3 |
|---|---|---|---|---|---|
| Gfortran Memory Using | 24.1 | 24.94mb | 24.1 | 24.9mb | 0 |
| Gfortran CPU Using | 25.2% | 25.1% | 25% | 25.19% | 0 |
| Ifortran Memory Using | 0 | 24mb | 24.9mb | 0 | 0 |
| Ifortran CPU Using | 0* | 25% | 25% | 0 | 0 |

* Zero refers to very close number to zero (it is not exactly equal zero). it might be because of the very fast completion time that we can not measure that.

***NOTE:** approximately in all cases, specifying an optimization level for completion boost program execution performance. On the other hand, higher levels of optimization rise completion time or may remarkably increase code size. For most cases, level -O3 is a good balance between performance gain, code size, and compilation time.

## 3 Method 2

In this method, we will use the OpenMP Library for parallel programming. The primary reason for parallel programming is to effectively execute code, as parallel programming saves time, allowing applications to be executed in a shorter period of time. we must add -fopenmp and -qopenmp for Ifort and Gfortran respectively.Following that, the command for Gfortran compiler is (Gfortran -o amin.x math.f90 fillMatrix.f90 matrix_subroutine.f90 -fopenmp ) and for Ifortran compiler is (ifort -o amin.x math.f90 fillMatrix.f90 matrix_subroutine.f90 -qopenmp). For using threads after compiling, we use this command: OMP_NUM_THREADS= n (n is a variable which takes value of 1,2 and 4.). Then, we run it again for example :(OMP_NUM_THREADS= 1 ./amin.x) In addition, based on the structure given in the assignment i checked different number threads (1,2 and 4).

The results are shown in the following table:

| Compiler | with OpenMP |
|---|---|
| Gfortran | 8.488 |
| Ifortran | 0.471 |

| Compiler | OMP_NUM_THREADS=1 | OMP_NUM_THREADS=2 | OMP_NUM_THREADS=4 |
|---|---|---|---|
| Gfortran | 9.015 | 8.741 | 8.841 |
| Ifortran | 0.489 | 0.464 | 0.47 |

As it can be seen, when we use OMP_NUM_THREADS =1 is slower than serial code.On the other hand, using threads 2 or 4 is going to be faster.

| Hardware | OpenMP |
|---|---|
| Gfortran Memory Using | 24.1 |
| Gfortran CPU Using | 25.13% |
| Ifortran Memory Using | 0 |
| Ifortran CPU Using | *0 |

* Zero refers to very close number to zero (it is not exactly equal zero). it might be because of the very fast completion time that we can not measure that.

## 4 Method 3

In this method, we use the BLAS library. Basic Linear Algebra Subprograms (BLAS) are a specification that prescribes a set of low-level routines for performing common linear algebra operations such as vector addition, scalar multiplication, dot products, linear combinations, and matrix multiplication. BLAS take advantage of the Fortran storage structure and the structure of the mathematical system wherever possible and most of the computers have blas library optimized to their system. Furthermore, we use a subroutine that name is matrix_blas.f90 and the command is : (gfortran/ifort -o matrix_blas.f90 fillMatrix.f90 -lblas) which it seems fast. BLAS subroutine calls are easier to code than the operations they replace and BLAS code is likely to perform well on all platforms and also are portable across different platforms.

| Compiler | with Blas |
|---|---|
| Gfortran | 0.550 |
| Ifortran | 0.454 |

| Hardware | BLAS Library |
|---|---|
| Gfortran Memory Using | 0 |
| Gfortran CPU Using | 0 |
| Ifortran Memory Using | 0 |
| Ifortran CPU Using | *0 |

* Zero refers to very close number to zero (it is not exactly equal zero). it might be because of the very fast completion time that we can not measure that.

## 5 Conclusion

All in all, we found that Ifort (intel) is faster than Gfortran in most of the experiments and it use less memory and CPU for matrix-matrix multiplication. blas library is optimized with most of the computers and this might be because ifortran and gfortran completion time is so close to each other in blas library. Parallel programming is not significantly fast in all situations and parallelize programs are difficult and time consuming.