Assignment 1

Amin Alimoradi

October 7, 2022

1 Introduction

This report illustrates a model that defines a horde of experiments on matrix-matrix multiplication with FORTRAN language. In this report, we will address the time wall and results with Ifort, Gfortran, parallel programming, and Basic Linear Algebra Subprograms (BLAS). We desire to time-wall 10 seconds during the process with the GNU compiler. After that, we will be testing the codes with different compilers to figure out them how much they use CPU and Memory for calculating matrix-matrix multiplication. Methods will test when the computer is not moving.

2 Methods

First of all, we write our code in a Text editor by explaining step by step. Math.f90 is our main project for creating matrix-matrix multiplication then we need a subroutine program for the structure of the matrix and call random numbers for them. we test our code with the Gfortran compiler by the command is: (Gfortran -o amin.x math.f90 fillMatrix.f90) then in next line we run the project : (./amin.x). The process takes approximately 10 seconds by Gfortran compiler. On the other hand, we desire to 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, In this case, we use more cores to calculate the program faster than the default. First, we use another command which concludes -O1, -O2, -O3, and -O4. Our command is this : (Gfortran -O1 -o math.f90 fillMatrix.f90) we should just change the number of -O. Also, we use these codes again for Ifort too.

The results show in the table:

Compiler	without OMP	-O0	-O1	-O2	-O3
Gfortran	10.19323900000	9.907668000000	7.884504000000	8.072606000000	4.164157000000
Ifortran	0.475549000000000	10.474358000	8.0858780000	0.47595200000	0.16470600000

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

3 Method 2

In addition, we want to use OpenMP Library to do parallel programming The main reason for parallel programming is to execute code efficiently, since parallel programming saves time, allowing the execution of applications in a shorter wall-clock time. Following that, we add matrix-subroutine.f90 in command. Also, we have to use a specific command for Ifort and Gfortran which is -fopenmp and -qopenmp respectively.Finally the command is (Gfortran -o amin.x math.f90 fillMatrix.f90 matrix-subroutine.f90 -fopenmp) and (ifort -o amin.x math.f90 fillMatrix.f90 matrix-subroutine.f90 -qopenmp) The results show on the table:

Compiler	with OpenMP
Gfortran	8.488888999999
Ifortran	0.47164900000

Hardware	BLAS Library
Gfortran Memory Using	24.1
Gfortran CPU Using	25.13%
Ifortran Memory Using	0
Ifortran CPU Using	0

4 Method 3

In this case, we want to 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. Furthermore, we use matrix_blas.f90 and fillMatrix.f90 that with this command: (gfortran/ifort -o matrix_blas.f90 fillMatrix.f90 -lblas) 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	0.5504249999999
Ifortran	0.45463700000

Hardware	BLAS Library
Gfortran Memory Using	0
Gfortran CPU Using	0
Ifortran Memory Using	0
Ifortran CPU Using	0

5 Conclusion

All in all, we found that Ifort (intel) is so faster than Gfortran in most of the experiment and it use less memory and CPU for matrix-matrix multiplication.