Information Theory and Computation
Exercise 3

Vincenzo Maria Schimmenti - 1204565

October 29, 2019

Theory

We rewrite a matrix multiplication program (either using different loop orders or the intrinsic Fortran function) with dynamic matrix sizing. We use a debug subroutine stored in a module for signaling possible errors or exceptions raised in the code.

Code Development

The matrix multiplication function has been implemented in 6 possible ways i.e. corresponding to the possible permutation in the loop order of the matrix indices in:

$$c(i,j) \leftarrow c(i,j) + a(i,k) \times b(k,j) \tag{1}$$

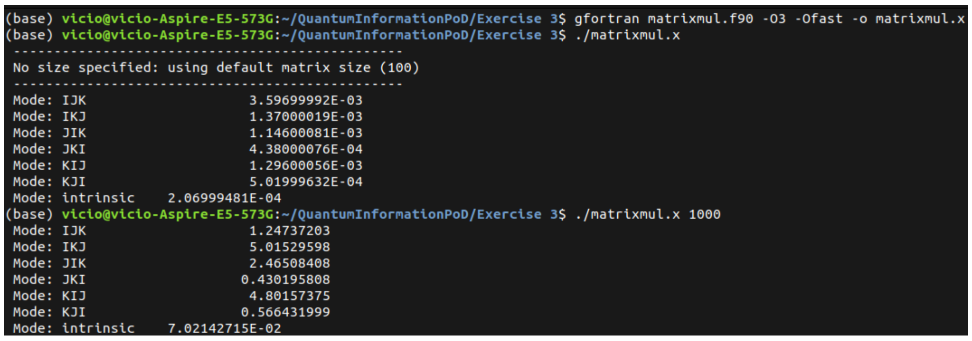
here a and b are input matrices and c is the resulting one; $\leftarrow$ is the assignment operator in pseudocode. The signature of the debug subroutine is the following:

```
SUBROUTINE DEBUGPROC(DEBUG,CONDITION,MSG,CONTENT,FRAME,STOPONERROR)
```

Here DEBUG, CONDITION,FRAME and STOPONERROR (with only the first one non-optional), MSG is a string and CONTENT is a generic scalar (an integer, a real or a logical). If DEBUG is TRUE we signal that we are in the debugging phase so in case of error we wish to get an error message; CONDITION is a logical telling whether we found an error or not: by default it is TRUE; FRAME is used to set a frame around the error message: by default is TRUE; STOPONERROR if TRUE stops the program in case of error (i.e. CONDITION is TRUE); MSG is the error message as a string; CONTENT is an optional scalar variable printed alongside the error message.

Results

Here we show two executions of the compiled program, one without any arguments which raises an exception signaled by the framed message and one with a size of 1000 which indeed does not raises any errors:



```
(base) vicio@vicio-Aspire-E5-573G:~/QuantumInformationPoD/Exercise 3$ gfortran matrixmul.f90 -O3 -Ofast -o matrixmul.x
(base) vicio@vicio-Aspire-E5-573G:~/QuantumInformationPoD/Exercise 3$ ./matrixmul.x
 -----------------------------------------------
 No size specified: using default matrix size (100)
 -----------------------------------------------
 Mode: IJK                  3.59699992E-03
 Mode: IKJ                  1.37000019E-03
 Mode: JIK                  1.14600081E-03
 Mode: JKI                  4.38000076E-04
 Mode: KIJ                  1.29600056E-03
 Mode: KJI                  5.01999632E-04
 Mode: intrinsic    2.06999481E-04
(base) vicio@vicio-Aspire-E5-573G:~/QuantumInformationPoD/Exercise 3$ ./matrixmul.x 1000
 Mode: IJK                  1.24737203
 Mode: IKJ                  5.01529598
 Mode: JIK                  2.46508408
 Mode: JKI                 0.430195808
 Mode: KIJ                  4.80157375
 Mode: KJI                 0.566431999
 Mode: intrinsic    7.02142715E-02
```

This is done by the following piece of code:

```
debugMode = .TRUE.
if(COMMAND_ARGUMENT_COUNT() < 1)then
        call DEBUGPROC(DEBUG=debugMode,MSG="No␣size␣specified:␣using␣default␣matrix␣size␣(100)")
        nn = 100
else
        call GET_COMMAND_ARGUMENT(1, nnchar)
        read(nnchar,FMT=*, IOSTAT=stat)nn
        if((stat.NE.0).OR.(nn <= 0))then
        call DEBUGPROC(DEBUG=debugMode,MSG="No␣size␣specified:␣using␣default␣matrix␣size␣(100)")
        nn = 100
        end if
endif
```

Here we check for no passed argument: if so we set the use the default matrix size; if an argument is passed we try to read it into an integer: if an error is raised - either signaled by IOSTAT argument of read or by matrix size being negative - we again use the default matrix size.

Self Evalutation

The main concern of the code was to write a good debug procedure in order to catch different possible scenarios: a good solution is provided by the generic argument CONTENT which can be of any scalar type. The debug subroutine can be further generalized in order to print vector and matrix types formatted in a readable way or to redirect the input towards a file.