Excercise 3

Abstract

This FORTRAN-90 project consists in the design of a subroutine to check and handle errors. This one will be applied to a simple code to compute the matrix multiplication to exploit its functionalities.

1 Theory

The matrix multiplication is a composition of linear transformation. Let

$$U : R^n \to R^p$$
$$T : R^p \to R^m$$

be transformations. Their composition is

$$\Phi = T \cdot U = T(U(x))$$
$$\Phi : R^n \to R^m$$

If U and T are linear transformation also their composition Φ will be linear. Hence all of them can be represented as matrices, and the elements φ_{ij} of Φ can be written as

$\varphi_{ij} = \sum_{k=1}^{P} t_{ik}u_{kj}$ where u_{kj} are the elements of the matrix U and t_{ik} are the elements of the matrix T.

2 Code Development

The code developed is stored in two different modules: **matmulqic** and **debugqic**.

Debugqic contains just the subroutine used for the debugging of the code, and it is called **breakifn**. This accepts as input three instances: A string **d_string**, and two logical variables (**cndtion** and **debug**). The functioning

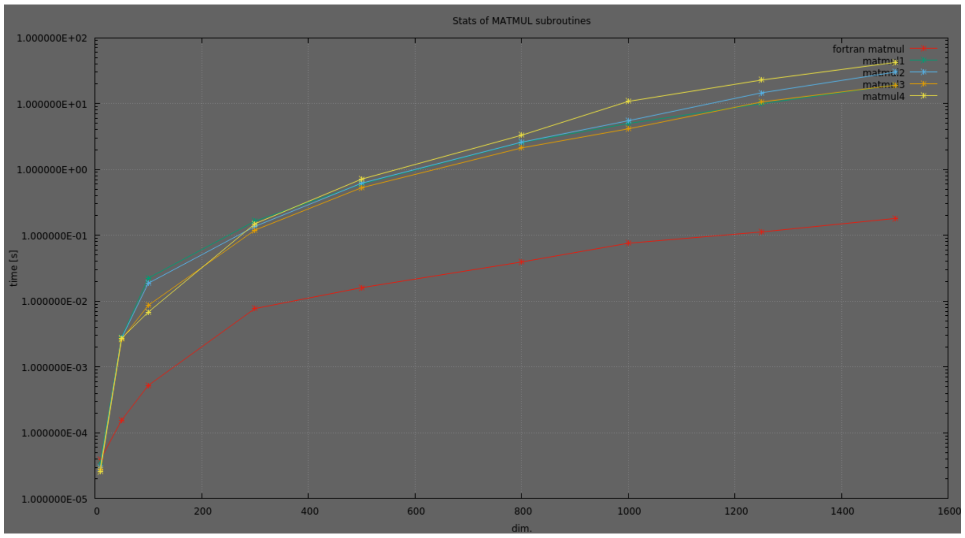

Figure 1: Plot showing the computation time registered of the various functions used without using compiler optimization. The tests have been conducted using as input two square matrices of dimension dim.×dim. The y axis is in log scale. The Fortran built-in function has a lower computation time with respect to the others, which are almost equivalent.

of this subroutine consists in two if statements which check for the two boolean variables, if the debug if enabled (**debug** is true) and the condition is not satisfied (**cndtion** is false) then the string is printed to screen and the execution of the program is terminated.

Listing 1: debugqic.f90 module

```
// debugqic.f90
module debugqic
contains
    subroutine breakifn(d_string, cndtion, debug)
        implicit none
        character(len = *) :: d_string
        logical :: cndtion, debug
        if (debug) then
            !if error occurs stop the program
            if (.not. cndtion) then
                write (*,*) d_string
                write(*,*) "Execution terminated"
                STOP
            end if
        end if
    end subroutine breakifn
end module debugqic
```

Matmulqic instead is a module, based on the previous debug environment, that allows to compute the matrix multiplication, generate random real matrix with elements in the range[0,1] and to print such matrices to screen. In the file are present four subroutines that computes the the matrix multiplication (**matmul1**,**matmul2**, **matmul3** and **matmul4**). These allocates dynamically memory to the result and solve this task using three nested for loops, with the only difference of interchanging the order of iteration of the rows of the first matrix with the columns of the second one, or computing the transpose of one of the input matrices to allow iteration just over rows or just over columns. The **brekifn** subroutine is applied to the matrix multiplication routines to check the equality between the columns of the left matrix in the product and the rows of the one on the right.

3 Results

The debug subroutine **breakifn** performed as expected in the code. If the debug variable was enabled and a false logical expression was used as input, the subroutine printed the respective message and terminated the program. This has been appreciated also while running the matrix multiplication code with incorrect inputs.

About the performances of the matrix multiplication codes I observed that the built-in fortran function computes much faster than the custom defined ones as we can see in [Fig.1]

4 Self Development

Applying a debug function as **breakifn** helps both the designer and the final user of the application achieving better error handling in the software.

Listing 2: Code of **matmul1** in matmulqic.f90

```
    // matmulqic.f90
    !perform matrix multiplication
    subroutine matmul1(aa,bb,result)
      implicit none
      integer*2 :: ii,jj,kk
      real, dimension(:,:), intent(in) :: aa,bb
      real, dimension(:,:), allocatable, intent(out) :: result

      integer, dimension(size(shape(aa))) :: sizea
      integer, dimension(size(shape(bb))) :: sizeb

      sizea=shape(aa)
      sizeb=shape(bb)

      allocate (result(sizea(1),sizeb(2)))

      call breakifn("invalid matrix shapes", (sizea(2) .eq.
          sizeb(1)), debug_var)
!-------------------------------------------------------
!part that will change in the other functions

        do ii=1,sizea(1)
          do jj=1,sizeb(2)
            result(ii,jj)=0
            do kk=1,sizea(2)
              result(ii,jj) = result(ii,jj)+ aa(ii,kk)*bb(kk,jj)
            end do
          end do
        end do

    end subroutine matmul1
```

Listing 3: Differences of matmul1 with matmul2, matmul3, matmul4 subroutines defined in matmulqic.f90

```
!from matmul2

        do jj=1,sizeb(2)
          do ii=1,sizea(1)
            result(ii,jj)=0
            do kk=1,sizea(2)
              result(ii,jj) = result(ii,jj)+ aa(ii,kk)*bb(kk,jj)
            end do
          end do
        end do

!------------------------------------------------------------

!from matmul3

        allocate(aa1(sizea(2), sizea(1)))
        aa1=transpose(aa)
        do jj=1,sizeb(2)
          do ii=1,sizea(1)
            result(ii,jj)=0
            do kk=1,sizea(2)
              result(ii,jj) = result(ii,jj)+ aa1(kk,ii)*bb(kk,jj)
            end do
          end do
        end do

!------------------------------------------------------------

!from matmul4

        allocate(bb1(sizeb(2), sizeb(1)))
        bb1=transpose(bb)
        do ii=1,sizea(1)
          do jj=1,sizeb(2)
            result(ii,jj)=0
            do kk=1,sizea(2)
              result(ii,jj) = result(ii,jj)+ aa(ii,kk)*bb1(jj,kk)
            end do
          end do
        end do
```