Trabalho 1 - Computação de Alto Desempenho - 2021.1
Nome: Carlos Henrique Ferreira Brito Filho
DRE: 120081409

1 Introdução

O trabalho 1 da disciplina Computação de Alto Desempenho consiste em implementar uma multiplicação matriz-vetor em *C* e em *Fortran* numa ordem de linha-coluna e depois numa ordem de coluna-linha, avaliando o tempo que levaria em todos os casos.

2 Estimando o tamanho máximo dos *arrays*

Para estimar o tamanho máximo dos *arrays* foi utilizado a seguinte fórmula:

$$(2x + x^2) * b = m * 10^9$$

Onde x será a dimensão dos vetores e da matriz, b vai ser a quantidade de *bytes* da variável utilizada na multiplicação e m é a quantidade de memória *RAM* do sistema. Para estimar o tamanho máximo que o sistema conseguiria utilizar foram definidos $m = 8$, pois o sistema utilizado possui 8GB de memória RAM, e $b = 8$, pois o tipo de variável a ser utilizada foi *double*. Logo, resolvendo a fórmula para x foi obtido $x \approx 30000$. Porém, devido aos processos do próprio sistema operacional, ao utilizar 30000 como dimensão dos vetores e da matriz o sistema operacional executa *kill* no processo. Então, através do método de tentativa e erro (o programa era executado para valores próximos de 30000) foi encontrado o valor $x = 27500$ para o máximo tamanho dos *arrays* pois a partir de tal valor o sistema operacional executaria *kill* no processo.

3 Implementação e execução do código

Em ambas as implementações (tanto em *C* quanto em *Fortran*) é executado um *loop* que incrementa o tamanho dos *arrays* que serão alocados na memória e preenchidos com valores aleatórios. Após o preenchimento é executada a multiplicação matriz-vetor numa ordem de linha-coluna e depois numa ordem de coluna-linha. Cada ordem de multiplicação é cronometrada e é gravada, junto do valor do tamanho dos *arrays*, num arquivo .csv. Depois que todos os dados foram coletados são geradas as curvas mostrando o tempo de execução utilizando *Python* e as bibliotecas *pandas* e *matplotlib*. Para facilitar a execução do código, foi criado um *script* auxiliar que instala as bibliotecas necessárias caso o usuário não as possua instaladas no sistema e executa o código automaticamente. A implementação do código pode ser encontrada no fim deste relatório e no seguinte repositório no GitHub: `https://github.com/carloshenriquefbf/HighPerformanceComputing/tree/main/exercise1`

4 Curvas mostrando o tempo de execução

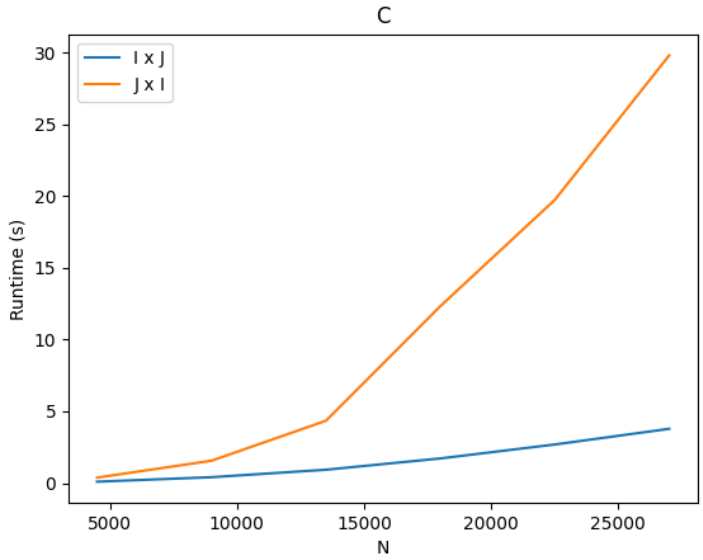

Figura 1: Gráfico para a implementação em C

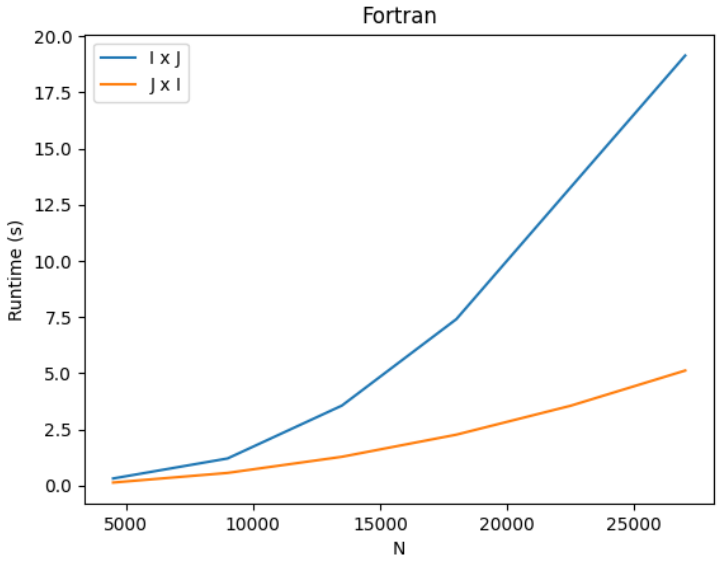

Figura 2: Gráfico para a implementação em Fortran

Como utilizamos múltiplas iterações aninhadas percorrendo os nossos dados é possível perceber a complexidade se aproximando da quadrática ($O(n^2)$).

5 A diferença no armazenamento de *arrays* em cada linguagem

Através dos gráficos do tópico anterior se torna perceptível que há diferença não apenas entre as linguagens mas também na ordem de execução do algoritmo de multiplicação. Como a linguagem *C* armazena *arrays* através do método ordem principal de linha, ou seja, elementos consecutivos de uma linha residem um ao lado do outro na memória, ela possuirá melhor desempenho quando o algoritmo de multiplicação utilizar a ordem linha-coluna (I x J). Alternativamente, a linguagem *Fortran* armazena *arrays* através do método ordem principal de coluna, ou seja, elementos consecutivos de uma coluna residem um ao lado do outro na memória. Logo, ela possuirá um melhor desempenho ao executar o algoritmo de multiplicação na ordem coluna-linha (J x I).

6 Códigos

6.1 Instruções

1. Clonar o repositório do *GitHub*:

   ```
   git clone https://github.com/carloshenriquefbf/
     HighPerformanceComputing.git
   ```

2. Para rodar o *script* de execução do trabalho 1 é necessário estar na pasta correta:

   ```
   cd /HighPerformanceComputing/exercise1
   ```

3. Transformar o *script* em exectuável

   ```
   chmod +x runner.sh
   ```

4. Rodar o *script* (ao rodar ele vai instalar, via *pip*, as bibliotecas *Python* necessárias para a plotagem dos gráficos então tenha certeza que o *pip* está instalado)

   ```
   ./runner.sh
   ```

   O *script* irá armazenar os dados e os gráficos mostrando o tempo de execução necessários para a execução das multiplicações matriz-vetor na pasta '*documents*'.

6.2 Bash

```
#!/bin/bash
echo 'Making sure the required python libraries are installed...'
pip install pandas
pip install matplotlib
echo 'Done!'
```

```
echo '------------------------------'

echo 'Compiling matrixVectorMultiplication on C...'
cd c
gcc matrixVectorMultiplication.c -o matrixVectorMultiplication
echo 'Done!'
echo 'Running matrixVectorMultiplication on C:'
./matrixVectorMultiplication
echo 'Done!'
echo 'The data is available on the documents folder'

echo '------------------------------'

echo 'Compiling matrixVectorMultiplication on Fortran...'
cd ../fortran
gfortran matrixVectorMultiplication.f95 -o
    matrixVectorMultiplication
echo 'Done!'
echo 'Running matrixVectorMultiplication on Fortran:'
./matrixVectorMultiplication
echo 'Done!'
echo 'The data is available on the documents folder'

echo '------------------------------'

echo 'Plotting graphs...'
cd ../python
python3 graph.py

echo '------------------------------'

echo 'Done!'
echo 'The graphs for each language are available on the documents
    folder'
```

6.3 C

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

double **fillMatrix(int size)
{
```

```
    double **matrix = (double **)malloc(size * sizeof(double *));

    for (int i = 0; i <= size; i++){
        matrix[i] = (double *)malloc(size * sizeof(double));
    }

    for (int i=0; i<size; i++) {
        for(int j=0; j<size; j++){
            matrix[i][j] = rand();
        }
    }
    return matrix;
}

void freeMatrix(double **matrix, int size)
{
    for (int i = 0; i <= size; i++){
        free(matrix[i]);
    }

    free(matrix);
}

double *fillVector(int size)
{
    double *vector = (double *)malloc(size * sizeof(double *));

    for (int i = 0; i < size; i++) {
        vector[i] = rand();
    }

    return vector;
}

double *fillResultIJ(int size)
{
    double *resultIJ = (double *)malloc(size * sizeof(double *));

    for (int i = 0; i < size; i++) {
        resultIJ[i] = 0;
    }

    return resultIJ;
}
```

```
double *fillResultJI(int size)
{
    double *resultJI = (double *)malloc(size * sizeof(double *));

    for (int i = 0; i < size; i++) {
        resultJI[i] = 0;
    }

    return resultJI;
}

double *matrixVectorMultiplicationIJ(double **matrix, double *
   vector, double *result, int size)
{
    for (int i = 0; i <= size; i++) {
        for (int j = 0; j <= size; j++) {
            result[i] = result[i] + (matrix[i][j] * vector[j]);
        }
    }
    return result;
}

double *matrixVectorMultiplicationJI(double **matrix, double *
   vector, double *result, int size)
{
    for (int j = 0; j <= size; j++) {
        for (int i = 0; i <= size; i++) {
            result[i] = result[i] + (matrix[i][j] * vector[j]);
        }
    }
    return result;
}

int main(void)
{
    srand(time(NULL));
    FILE *ij, *ji;
    clock_t start_t, end_t;
    double total_t;
    int size;

    ij = fopen("../documents/c/csv/matrixVectorMultiplicationIJ.
       csv", "w+");
    ji = fopen("../documents/c/csv/matrixVectorMultiplicationJI.
       csv", "w+");
```

```
    for(int i=1;i<=6;i++)
    {
        size = 4500 * i;
        double **matrix = fillMatrix(size);
        double *vector = fillVector(size);

        double *resultIJ = fillResultIJ(size);
        start_t = clock();
        resultIJ = matrixVectorMultiplicationIJ(matrix, vector,
           resultIJ, size);
        end_t = clock();
        total_t = ((double)(end_t - start_t)) / CLOCKS_PER_SEC;
        printf("\nIt took the computer %.6f s, to compute a %d
           degree matrix on IJ; i = %d\n", total_t, size,i);
        fprintf(ij, "%d,%.6f\n", size, total_t);
        free(resultIJ);

        double *resultJI = fillResultJI(size);
        start_t = clock();
        resultJI = matrixVectorMultiplicationJI(matrix, vector,
           resultJI, size);
        end_t = clock();
        total_t = ((double)(end_t - start_t)) / CLOCKS_PER_SEC;
        printf("\nIt took the computer %.6f s, to compute a %d
           degree matrix on JI; i = %d\n", total_t, size,i);
        fprintf(ji, "%d,%.6f\n", size, total_t);
        free(resultJI);

        freeMatrix(matrix, size);
        free(vector);
    }

    fclose(ij);
    fclose(ji);

    return 0;
}
```

6.4 Fortran

```
program matrixVectorMultiplication
implicit none
```

```
integer size, n
real*8, dimension(:), allocatable :: vector, resultIJ, resultJI
real*8, dimension(:,:), allocatable :: matrix
real start_t, end_t, total_t
call random_seed()

open(1, file = '../documents/fortran/csv/
   matrixVectorMultiplicationIJ.csv', action = 'write')
open(2, file = '../documents/fortran/csv/
   matrixVectorMultiplicationJI.csv', action = 'write')

do n = 1, 6, 1
    size = 4500 * n
    allocate(vector(size))
    allocate(matrix(size,size))

    call fillVector(vector,size)
    call fillMatrix(matrix,size)

    allocate(resultIJ(size))
    call fillResultsVector(resultIJ,size)

    call cpu_time(start_t)
    call matrixVectorMultiplicationIJ(matrix,vector,resultIJ,size)
    call cpu_time(end_t)

    total_t = end_t - start_t
    print *, 'It took the computer', total_t, 'to compute a ',size
       ,'degree matrix on IJ; i = ',n
    write(1,*) size,",",total_t
    deallocate(resultIJ)

    allocate(resultJI(size))
    call fillResultsVector(resultJI,size)

    call cpu_time(start_t )
    call matrixVectorMultiplicationJI(matrix,vector,resultJI,size)
    call cpu_time(end_t )

    total_t = end_t - start_t
    print *, "It took the computer", total_t, "to compute a ",size
       ,"degree matrix on JI; i = ",n
    write(2,*) size,",",total_t
    deallocate(resultJI)
```

```
    deallocate(matrix)
    deallocate(vector)
end do

close(1)
close(2)

contains

subroutine fillVector(vector, size)
    implicit none

    real*8, dimension(:) :: vector
    integer :: size, i
    real*8 :: u

    do i = 1, size
            call random_number(u)
            vector(i) = u
    end do
end

subroutine fillResultsVector(vector, size)
    implicit none

    real*8, dimension(:) :: vector
    integer :: size, i

    do i = 1, size
            vector(i) = 0
    end do
end

subroutine fillMatrix(matrix, size)
    implicit none

    real*8, dimension(:,:) :: matrix
    integer :: size, i, j
    real*8 :: u

    do i = 1, size
        do j = 1, size
            call random_number(u)
            matrix(i, j) = u
        end do
```

```
    end do
end

subroutine matrixVectorMultiplicationIJ(matrix, vector, result,
   size)
    implicit none

    real*8, dimension(:,:) :: matrix
    real*8, dimension(:) :: vector, result
    integer :: size, i, j

    do i = 1, size
        do j = 1, size
            result(i) = result(i) + (matrix(i, j) * vector(j))
        end do
    end do
end

subroutine matrixVectorMultiplicationJI(matrix, vector, result,
   size)
    implicit none

    real*8, dimension(:,:) :: matrix
    real*8, dimension(:) :: vector, result
    integer :: size, i, j

    do j = 1, size
        do i = 1, size
            result(i) = result(i) + (matrix(i, j) * vector(j))
        end do
    end do
end

end program matrixVectorMultiplication
```

6.5 Python

```
import pandas as pd
import matplotlib.pyplot as plt

C_IJ = pd.read_csv("../documents/c/csv/
   matrixVectorMultiplicationIJ.csv",header = None)
C_JI = pd.read_csv("../documents/c/csv/
   matrixVectorMultiplicationJI.csv",header = None)
```

```
plt.figure(1)
plt.plot(C_IJ[0], C_IJ[1], label='I x J')
plt.plot(C_JI[0], C_JI[1], label='J x I')
plt.title('C')
plt.ylabel('Runtime (s)')
plt.xlabel('N')
plt.legend()
plt.savefig('../documents/c/images/graph.png')

FORTRAN_IJ = pd.read_csv("../documents/fortran/csv/
    matrixVectorMultiplicationIJ.csv",header = None)
FORTRAN_JI = pd.read_csv("../documents/fortran/csv/
    matrixVectorMultiplicationJI.csv",header = None)

plt.figure(2)
plt.plot(FORTRAN_IJ[0], FORTRAN_IJ[1], label='I x J')
plt.plot(FORTRAN_JI[0], FORTRAN_JI[1], label='J x I')
plt.title('Fortran')
plt.ylabel('Runtime (s)')
plt.xlabel('N')
plt.legend()
plt.savefig('../documents/fortran/images/graph.png')
```