LAB 2

DUE DATE: Oct 11 (upload to polylearn)

Name: Dhruv Singal

Write a program in C to perform Matrix Matrix multiplication. The requirements for the program:

1. You read from a file the values for the input matrix, write also the result of the matrix multiplication to a file.
2. Create your own test matrices with float random values range 0-100
3. Do not assume the size of the matrices are squared, matrix may be of any size and shape (for now is OK to assume they do fit in memory)
4. Matrix multiplication can be done very simple as shown in Fig 1, this algorithm is $O(n^3)$, and this algorithm is enough for this lab. NOTE: MAKE SURE YOUR ALGORITHM COMPILES AND RUNS ON THE LAB COMPUTERS.
   If you are curious and want 10 extra credit points you can also try to implement Strassen algorithm $O(n^{2.807})$, although to really see the speed up using Strassen your matrix must be at least n>100, these algorithm appears in several libraries (BLAS, etc).
   Matrix multiplication is still an area of research, most important problems do require multiplication -linear transformations for image/video processing, solving systems of linear equations, rank algorithms,...- of huge matrices (don't even fit in memory) and getting an algorithm that perform faster is paramount.

If

$$\mathbf{A} = \begin{pmatrix} a & b & c \\ x & y & z \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} \alpha & \rho \\ \beta & \sigma \\ \gamma & \tau \end{pmatrix},$$

their matrix products are:

$$\mathbf{AB} = \begin{pmatrix} a & b & c \\ x & y & z \end{pmatrix} \begin{pmatrix} \alpha & \rho \\ \beta & \sigma \\ \gamma & \tau \end{pmatrix} = \begin{pmatrix} a\alpha + b\beta + c\gamma & a\rho + b\sigma + c\tau \\ x\alpha + y\beta + z\gamma & x\rho + y\sigma + z\tau \end{pmatrix}$$

```
for (row=0; row<Hight; row++){
    for(col<0; col<Width; col++) {
        for(k=0; k<Width; k++){
              mm[row*Width+col]+=
              m1[K*Width+col]*m2[row*Width+k];
      }
    }
}

Where first matrix is rows1, cols1 columns and
second matrix is rows2 rows  and cols2 columns
```

Fig 1. Matrix Multiplication

What to turn in

Upload a report in PDF with the following:

1. Your name
2. Explanation of the matrix multiplication code used if different than the one provided in Fig 1
3. Table with execution times (only for the matrix multiplication, do not add the reading and writing of the matrix from/to a file)

Matrix A and B Size	Execution Time
A.first(100*100), second (100,100)	0.011
B. first(1000*300), second (300,700)	0.824
C. first(600*400), second (400,1000)	1.015
D. first(900*100), second (100,9000)	3.029

4. Table with the execution time for the multithreaded matrix multiplication algorithm

Matrix A and B Size	Execution Time 2 Threads	Execution Time 4 Threads	Execution Time 8 Threads
A.	0.007	0.003	0.002
B.	0.512	0.270	0.152
C.	0.643	0.336	0.188
D.	1.809	1.012	0.573

5. Appendix with your code, clean and properly commented Just the Matrix Multiplication function code(I may ask for a demo of your code working on the computer labs if I don't understand how your code is able to run).

```
void matrix_mult_nonthreaded(){
        int i,j,k;
        for(i=0;i<A_HEIGHT;i++){
                for(j=0;j<B_WIDTH;j++){
                        for(k=0;k<AB_SHARED;k++){
                                C[i][j]+=A[i][k]*B[k][i];
                        }
                }
        }
        return;
}

void* matrix_mult_threaded(void* id){
        int threadID=(int)id;
        int i,j,k;
        int num_rows_per_thread=A_HEIGHT/threads_available;
        int leftover_rows=A_HEIGHT%threads_available;
        int start_row=threadID*num_rows_per_thread;
        int stop_row=start_row+num_rows_per_thread;
        if(threadID==threads_available-1)
                stop_row=stop_row+leftover_rows;
        for(i=start_row;i<stop_row;i++){
                for(j=0;j<B_WIDTH;j++){
                        for(k=0;k<AB_SHARED;k++){
                                C[i][j]+=A[i][k]*B[k][i];
                        }
                }
        }
        return NULL;
}
```