**DECLARATION**

The submitted work (OpenMP program and Word file) are solely the work of the student (**Joseph Cherian**) except for elements that are clearly identified, cited and attributed to other sources.

**A description of the design of the program**

void populate\_matrix(int n){

//different seed for each trial

[1]

//populate matrices

for (int i = 0; i < n; i++){

for (int j = 0; j < n; j++){

matrix1[i][j] = rand();

matrix2[i][j] = rand();

}}}

**populate\_matrix** method is used to populate the matrices matrix1 and matrix2 by using two for loops as shown above. The rand function defined in the stdlib library is used to populate the matrices using the random number generator. The srand() function controls the choice of the seed by making use of the computer’s internal clock. [1]

void transpose(int n) {

int i,j;

for(i=0; i<n; i++) {

for(j=0; j<n; j++) {

transpose2[i][j] = matrix2[j][i];

} }}

**transpose** function is used to obtain the transpose of the second matrix used for multiplication(matrix 2).The resultant transpose matrix is stored in a new matrix transpose2.

double serial\_mult(int n){

int i, j, k;

//start time from the wall clock

double starttime = omp\_get\_wtime();

for (i = 0; i < n; i++){

for (j = 0; j < n; j++){

matrix3[i][j] = 0;

for (k = 0; k < n; k++){

matrix3[i][j] = matrix3[i][j] + matrix1[i][k] \* matrix2[k][j];

}}}

// end time from the wall clock

double endtime = omp\_get\_wtime();

//return the execution time

return endtime - starttime;

}

**serial\_mult** function is used to perform the matrix multiplication using the conventional method. The matrix multiplication is performed by multiplying each row element of matrix1 with all the column elements of matrix2 using 3 for loops. The execution time for the serial matrix multiplication is obtained with the help of the **omp\_get\_wtime** function which returns the start time and end time of the matrix multiplication execution. The endtime – starttime value is returned.

double optimized\_multiplication(int n){

int i, j, k;

//transpose of matrix 2

transpose(n);

//start time from the wall clock

double starttime = omp\_get\_wtime();

#pragma omp parallel for shared(matrix1, matrix2, matrix3) private(i, j, k) schedule(static) num\_threads(THREADS)

for (i = 0; i < n; i++){

for (j = 0; j < n; j++){

double sum = 0;

for (k = 0; k < n; k++){

sum += matrix1[i][k] \* transpose2[j][k];

}

matrix3[i][j] = sum;

}}

//get the end time from wall clock

double endtime = omp\_get\_wtime();

//return the execution time

return (endtime - starttime);

}

**optimized\_multiplication** function is used to perform the matrix multiplication using Openmp by taking the transpose of the matrix 2 to be multiplied. The execution time for the **optimized** matrix multiplication is obtained with the help of the **omp\_get\_wtime** function which returns the start time and end time of the matrix multiplication execution. The endtime – starttime value is returned. The matrix multiplication is performed by taking the sum of each of the row elements of matrix 1 with all the row elements of the transpose of matrix 2.

**#pragma omp parallel for shared(matrix1, matrix2, matrix3) private(i, j, k) schedule(static) num\_threads(THREADS)** is used to specify the variable scoping and to request threads for parallel execution by the various cores of the processor.

void time\_serial(){

for (int n = sample; n <= 1000; n = n + 100) {

printf("\nStarting for n = %d\n\n", n);

double sum = 0;

for (int i = 0; i < sample; i++){

populate\_matrix(n);

sum += serial\_mult(n);

}

double average = sum / sample;

printf("Serial: Time taken for a %d x %d matrix is %f s\n", n, n, average);

printf("\n---------------------------------------------\n");

}}

**time\_serial** function returns the time taken for the conventional matrix multiplication for various matrix sizes starting from the size given by the user. The size of the matrix is incremented by 100 each time. The matrix is populated using the **populate\_matrix** method as mentioned earlier. The average time taken for execution is calculated by executing the matrix multiplication for the sample size number of times which is done by using a for loop.

void time\_parallel(){

for (int n = sample; n <= 1000; n = n + 100){

printf("\nStarting for n = %d\n\n", n);

double sum = 0;

for (int i = 0; i < sample; i++){

populate\_matrix(n);

sum += optimized\_multiplication(n);}

double average = sum / sample;

printf("Optimized: Time taken for a %d x %d matrix is %f s\n", n, n, average); printf("\n---------------------------------------------\n");

}}

**time\_parallel** function returns the time taken for the optimised matrix multiplication using OpenMP for various matrix sizes starting from the size given by the user. The size of the matrix is incremented by 100 each time. The matrix is populated using the **populate\_matrix** method as mentioned earlier. The average time taken for execution is calculated by executing the matrix multiplication for the sample size number of times which is done by using a for loop.

int main()

{

char method;

//Enter method that needs to be run

printf("Enter one of the options given below\n \ts - for serial method \n\to - for optimized Openmp method\n");

scanf("%c", &method);

//Enter sample size for the selected method

printf("Enter sample size from 100: ");

scanf("%d", &sample);

switch (method) {

case 's':

time\_serial();

break;

case 'o':

time\_parallel();

break;

}

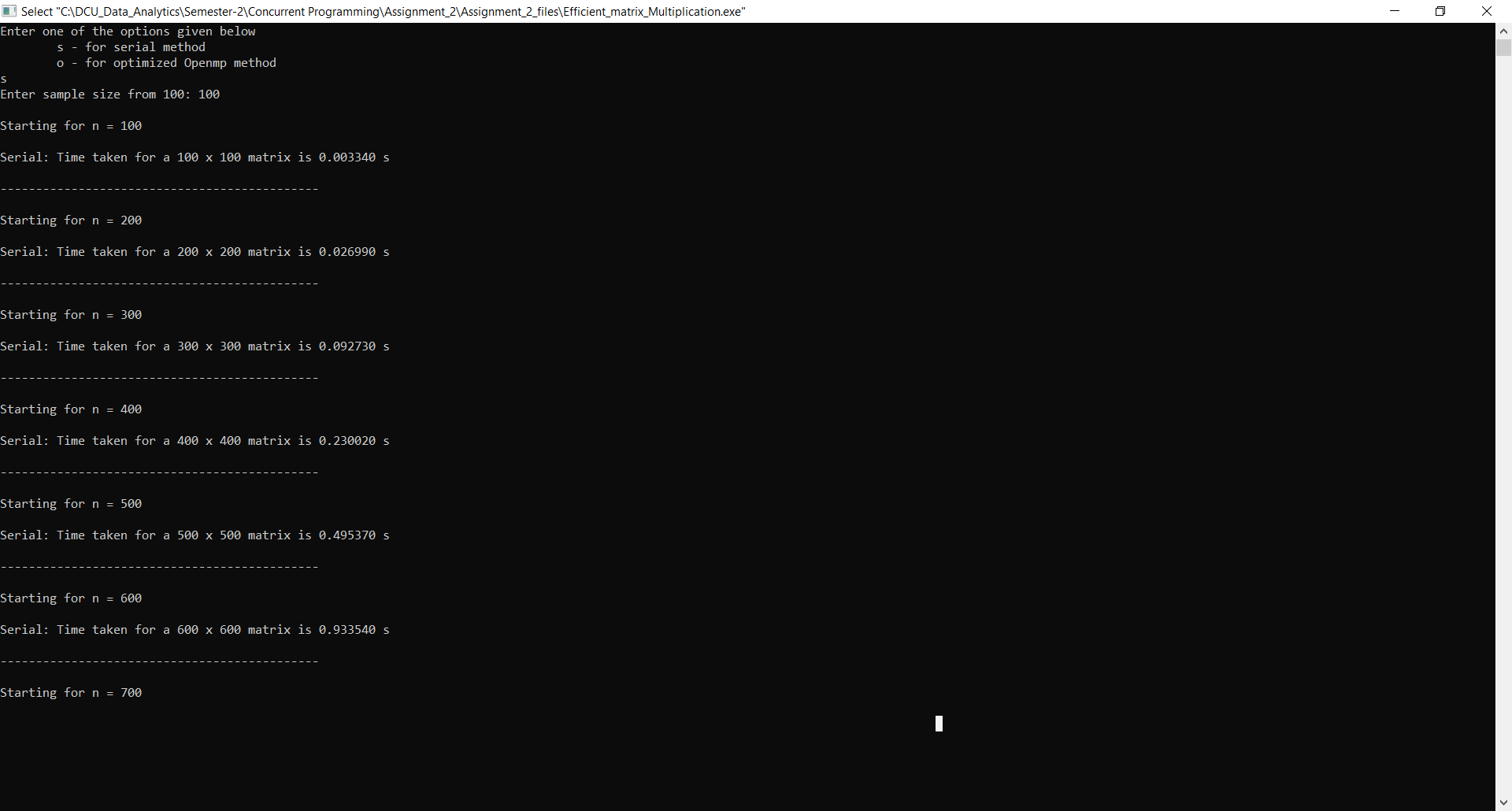
return 0;

}

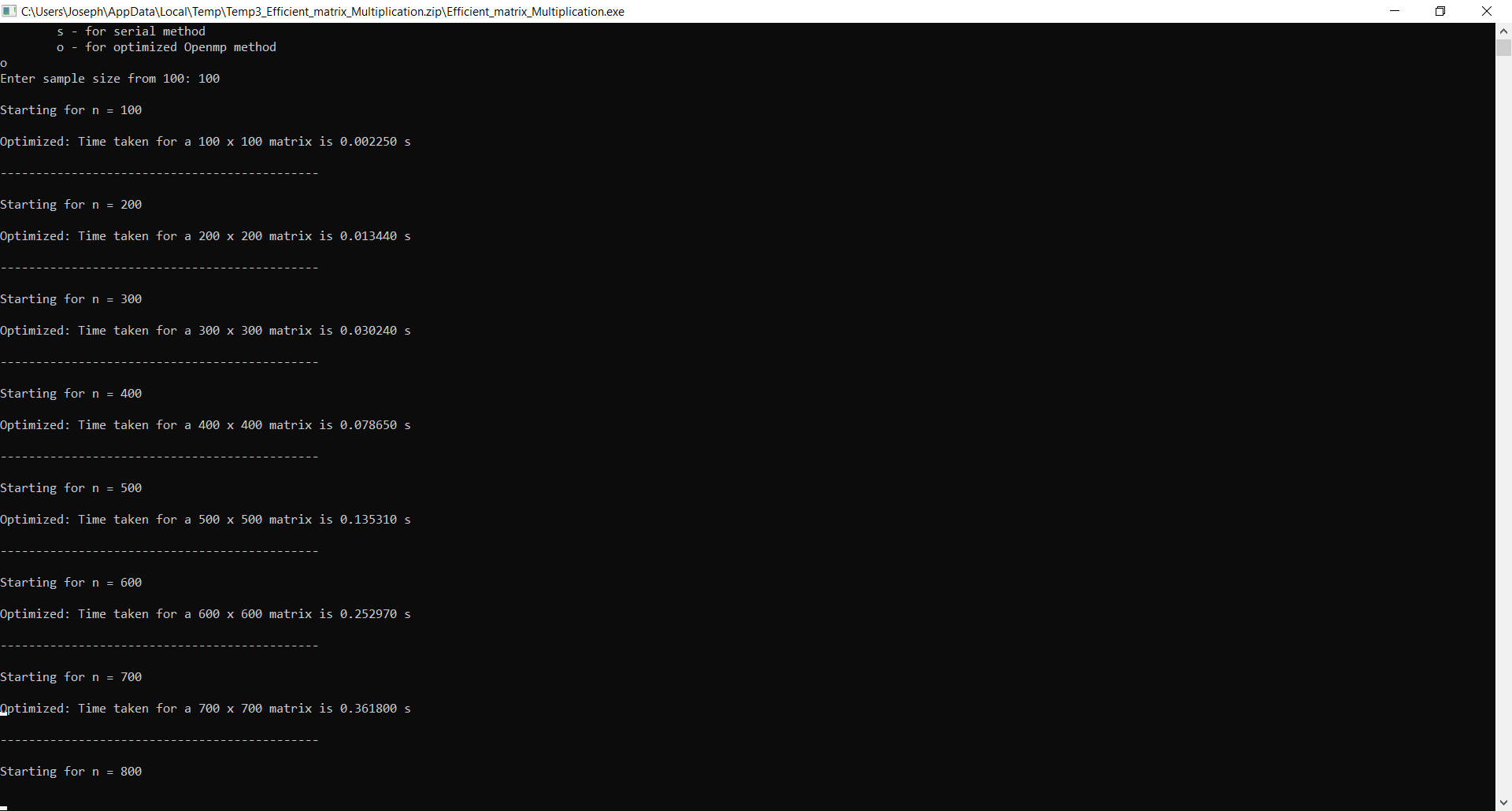
**main** method- The main method returns the time taken for the execution of matrix multiplication using either the conventional or optimized openmp method depending on the choice of the user. Once the choice of method for matrix multiplication and the sample size is selected by the user the time taken for the chosen matrix multiplication for the consecutive values (sample size, sample size+100….) is returned by using the switch case.

**Evidence as to the efficiency of implementation**

Time taken for Serial matrix multiplication (Conventional matrix multiplication)



Time taken for Optimized Openmp matrix multiplication



Time taken for the execution of matrix multiplication for the matrix sizes from 100 for both serial matrix multiplication and optimized OpenMP matrix multiplication is shown above.

The time taken for matrix size 100\*100 in serial matrix multiplication is 0.003340 s.

The time taken for matrix size 100\*100 in serial matrix multiplication is 0.002250 s.

The time taken for matrix size 200\*200 in Optimized OpenMP matrix multiplication is 0.026990 s.

The time taken for matrix size 200\*200 in Optimized OpenMP matrix multiplication is 0.013440 s.

The time taken for matrix multiplication for large matrices is comparatively lesser for the optimized OpenMP method than the conventional method as shown above.

Hence the matrix multiplication using the optimized OpenMP method by taking the transpose of the matrix is more efficient than the conventional matrix multiplication for **large** matrices.

The transpose method creates the transpose of **Matrix 2** in a buffer. This method gives the fastest result (matrix multiplication goes as O(n^3) and transpose as O(n^2) so doing the transpose is at least 1000x faster). The blocking method is slower. Another problem with blocking is it has to update the block several times. This is a challenge for threading/OpenMP because it can cause race conditions if one is not careful.[4]

# **References**

|  |  |
| --- | --- |
| [1] | "MathBits," [Online]. Available: https://mathbits.com/MathBits/CompSci/LibraryFunc/rand.htm. |
| [2] | D. C. a. R. B. HARI SINGH, "PARALLEL COMPUTING OF MATRIX MULTIPLICATION IN OPEN MP SUPPORTED CODEBLOCKS," *Advances and Applications in Mathematical Sciences,* 2019. |
| [3] | A. A. H. A. A. Khaled M. Matrouk, "Analysis of Matrix Multiplication Computational Methods," *European Journal of Scientific Research,* 2014. |
| [4] | "Stack Overflow," [Online]. Available: https://stackoverflow.com/questions/15829223/loop-tiling-blocking-for-large-dense-matrix-multiplication. |
| [5] | P. D. Michailidis, "Performance Models for Matrix Computations on Multicore Processors Using OpenMP," 2010. |