IRIS NICOLE CARSON 4 BS CpE March 20, 2023
ANTONIO RAFAEL CASTRO
JOSHUA MANUEL LOUISE KEMPIS

Project 2 Final Report **EARLY SUBMISSION**

Matrix Multiplication Speed Test

Overview

A speedtest review program will be created that would compare the computing speed between different matrix multiplication implementations in a C++ program depending on their use of threads. The program requests for two inputs M and N which will determine the overall size of the two matrices that will be multiplied to each other.

Code Collaboration Overview

The code is collaborated between three members which contribute to the common GitHub repository. This setup allows the members to update the code simultaneously and allows easy updates for code revisions between each member. This helps ensure a steady implementation process and allows an easy way to fix the common code run by each member of their individual computers.

Implementation of Functions and Operations

randNumb()

The randNumb() function provides randomly generated numbers to the two matrices provided their M and N sizes. These value generators are what will be used when performing the multiplication operation between the two matrices.

```
void randNumb(int row, int col, int **array, int seed)
{
  srand((unsigned) seed);

  // Allocate Random Generated Values to Array
  for (int i = 0; i < row; i++)
  {
    for (int j = 0; j < col; j++)
    {
      array[i][j] = rand() % 100; //range [0, 100]
    }
  }
}
```

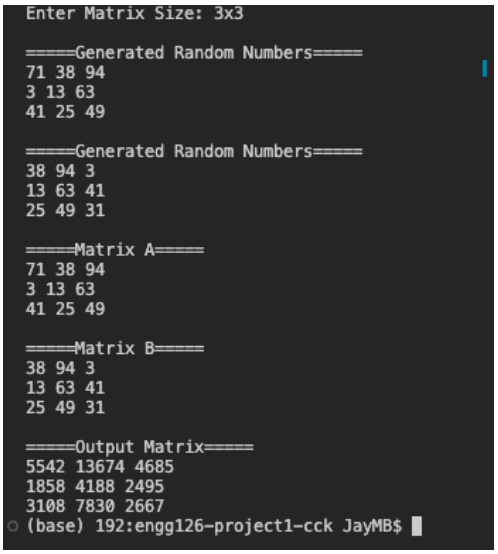
matrixMult(), matrixMult_row(), matrixMult_prod()

These functions are variations of the matrixMult() function that was initially implemented. This performs the basic matrix multiplication operation given the values from the variables that hold the data of the generated matrices.

```
void matrixMult(int Msize, int Nsize, int **A, int **B, int **C)
{
  // Initializing Elements of Array C to 0
  for(int i = 0; i < Msize; ++i)
  {
    for(int j = 0; j < Msize; ++j)
    {
      C[i][j]=0;
    }
  }

  // Multiplying Arrays A and B and Storing the Results in Array C
  for(int i = 0; i < Msize; ++i)
  {
    for(int j = 0; j < Msize; ++j)
    {
      for(int k = 0; k < Nsize; ++k)
      {
        C[i][j] += A[i][k] * B[k][j];
      }
    }
  }
}
```

Output Review

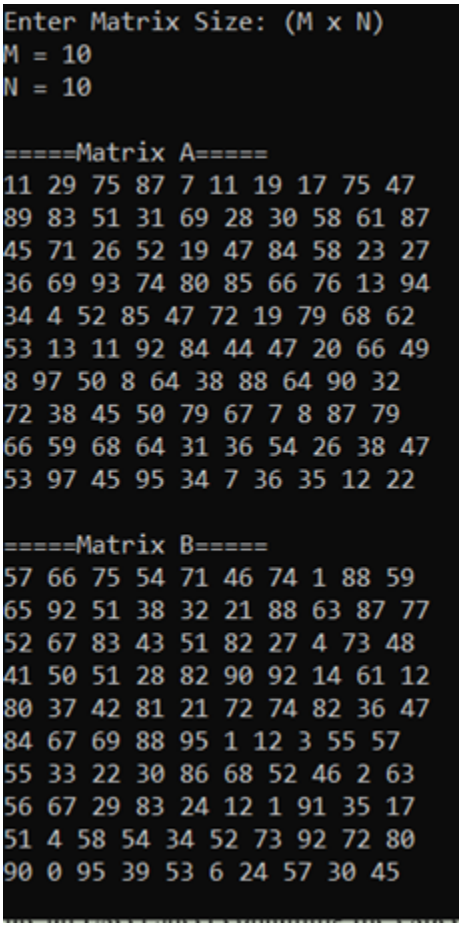


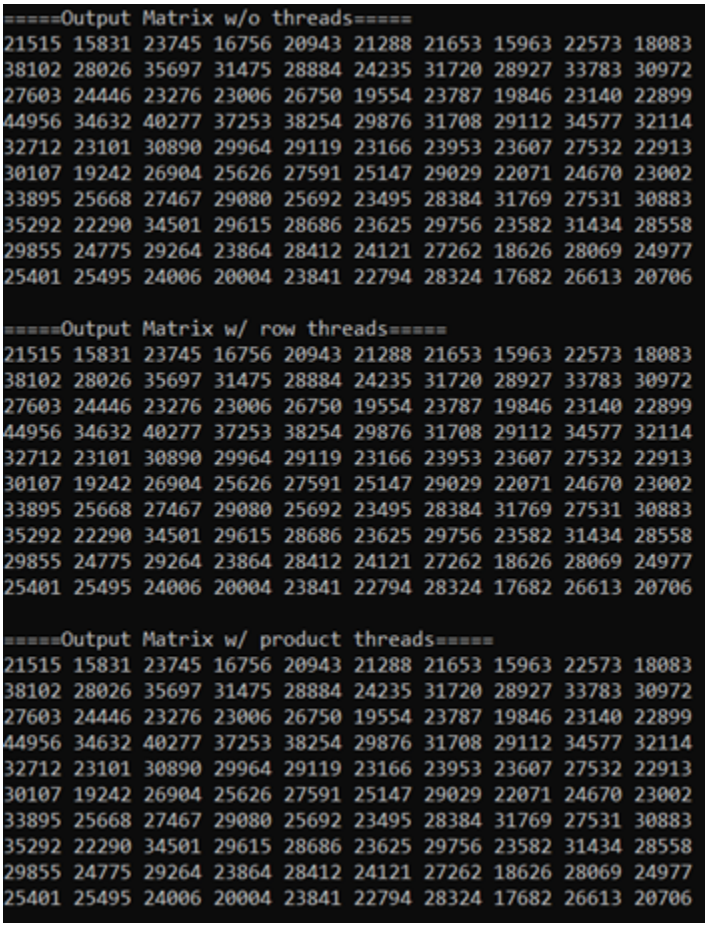
The program requests for a matrix size that would be used for the multiplication process. Two matrices are generated with randomly generated elements corresponding to the determined size. The input matrix dimensions reflect the dimensions for matrix A [MxN] and matrix B reflects the reversed dimension [NxM]. In the case where both the 'N' and 'M' input are the same number, it is recognized by the program as a square matrix.

The output matrix contains the values of the output of the multiplication operation between matrix A and matrix B

Implementation of Threads and Comparing Speed results

The function to implement threads in the multiplication process of the matrices has been implemented using the threads_row() and threads_prod() functions. This goes with the implementation of the multiplication operation functions matrixMult_row() and matrixMult_prod() which came from the original matrixMult() function. The results of the current code contains these output screenshots:





Given 10 trials with a duration of 1s each, the number of Matrix Multiplication operations were tested and the output for the Minimum, Maximum, Mean, and Variance results were provided.

These data is in relation to the speed at which the program was able to perform the multiplication operation following the implementation of the new matrixMult functions as well as the speedTest() function

```
Number of performed Matrix Multiplication in 10 trials with
duration of 1s each
>> w/o threads
Minimum = 157210
Maximum = 184424
Average = 169083
Variance = 8.80715e+07

>> w/ threads per row
Minimum = 866
Maximum = 952
Average = 931.5
Variance = 604.05

>> w/ threads per product
Minimum = 83
Maximum = 99
Average = 93.9
Variance = 24.89
```

Additional Trials

Additional trials were performed which attempted to run the program with larger matrix sizes and compare the final results between systems.

```
Number of performed Matrix Multiplication in 10 trials with duration of 1s each
>> w/o threads
Minimum = 397022
Maximum = 437559
Average = 429585
Variance = 1.50805e+08

>> w/ threads per row
Minimum = 12055
Maximum = 12743
Average = 12407.5
Variance = 48770.2

>> w/ threads per product
Minimum = 808
Maximum = 1340
Average = 1223.4
Variance = 21048.8
```

The result of running a 10x10 matrix size on an M1 system arrived at different values compared to running them on a standard intel processor.

```
Number of performed Matrix Multiplication in 10 trials with duration of 1s each
>> w/o threads
Minimum = 464
Maximum = 478
Average = 474.3
Variance = 14.81

>> w/ threads per row
Minimum = 731
Maximum = 848
Average = 820.9
Variance = 998.29

>> w/ threads per product
Minimum = 2
Maximum = 3
Average = 2.1
Variance = 0.09
```

The values for the Max, Min, Average, and Variance continue to differ upon the results of a matrix multiplication operation of size 100x100. The results are significantly smaller which indicate that over the 10 trials with duration of 1s, fewer multiplication operations were performed

Conclusion

As the matrix size gets bigger and bigger, it is possible to start seeing a change in which multiplication method produces more Matrix Multiplication operations over a given period of time. The second instance of the program screenshot (under *Additional trials*) which used a matrix size of 100x100 resulted in a larger number of performed operations under "w/ threads per row" with an average of 820.9 compared to "w/o threads" which averaged at 474.3. This is a stark contrast to that of the 10x10 matrix operation where "w/o threads' averaged at 429685 over the duration of 10 1-sec trials compared to "w/ threads per row" which averaged at 12407.5. In both larger and smaller sizes, using the "w/ threads per product" multiplication method, results in significantly lower min, max, and average output results.

Documentation Link

The GitHub repository for the project is made publicly available through the following:

GitHub Link	https://github.com/jaykempis/engg126-projects-cck/tree/main/Project2

References

JavaTPoint (2021) Matrix multiplication in C++ retrieved from: https://www.javatpoint.com/matrix-multiplication-in-cpp

@shubham_rana_77 (2023) Multiplication of Matrix using threads. GeeksforGeeks. Retrieved from: https://www.geeksforgeeks.org/multiplication-of-matrix-using-threads/