University of the Witwatersrand
School of Electrical & Information Engineering

ELEN4020A
Data Intensive Computing for Data Science
Laboratory 1 Report

Milliscent Mufunda - 1473979
Keanu Naidoo - 1422973
Matthew Woohead - 1385565

Due Date: 25 February 2019

***Abstract*—This report discusses the laboratory work done on multidimensional square matrix addition and multiplication. Algorithms that compute the array addition $C = A + B$ and multiplication $C = A \times B$ where A, B and C are either 2-dimensional (2-D) or 3-dimensional (3-D) square arrays are developed. The corresponding pseudo code is included. Along with the algorithms a main program that dynamically generates the elements of the input arrays is developed. The elements are randomly generated between [0, 20]. The algorithm is tested, critically analyzed and deemed fully functional.**

I. Introduction

Multidimensional matrix mathematics is a relatively new branch of mathematics with various applications in engineering, natural science, math and social sciences just to name a few [1]. This report discusses the development of algorithms for multidimensional matrix addition and multiplication in two multi dimensions; namely 2-D and 3-D. Included are the lab requirements and the pseudo code for the 2-D and 3-D matrix addition and multiplication algorithms. These can be found in sections II and III respectively. The algorithms' efficiency is also discussed in section IV. Lastly a method for extending the 3-D matrix multiplication algorithm such that it works with matrices with N dimensions is also discussed. This is discussed in section V.

II. Laboratory Requirements

The laboratory has the following requirements; it should be conducted using the Linux programming environment and with the use of one or more of specified editors. All laboratory documentation is to be done using Latex. The use of GIT and GitHub is also required. The problem to be addressed is the development of four separate procedures which are to handle the addition and multiplication of 2-D and 3-D matrices. The input matrices are to be sized 10×10 and then 20×20. These procedures are to be named:

- rank2TensorAdd()
- rank2TensorMult()
- rank3TensorAdd()
- rank3TensorMult()

To meet the requirements, all the coding is done using Ubuntu 16.04 and with the gedit editor. The laboratory documentation is done using Overleaf. A GIT repository is also used to track the history of the laboratory work done.

III. Explanation of Algorithms

The algorithms make use of conventional matrix addition and multiplication procedures.

A. Addition

The same algorithm is used for both 2D and 3D matrix addition. The matrix addition is done by simply adding corresponding matrix components with each other. For example, the first entry in matrix A and the first entry in matrix B will be added to produce the first entry in matrix C. This process is repeated until all entries in both matrix A and B have been summated, in order to produce matrix C. The pseudo code for 2D and 3D addition can be seen below in Algorithm 1 and 2.

```
Input: matrix A (N x N), matrix B (N x N)
Output: matrix C (N x N)
initialization matrix C;
for $i$ = $0$ to $N$ do
    initialization matrix Temp;
    for $j$ = $0$ to $N$ do
        Temp$_j \leftarrow A_{ij} + B_{ij}$
    end
    C$_i \leftarrow Temp_j$
end
```

Algorithm 1: Rank 2 Tensor Add

```
Input: matrix A (N x N x N), matrix B (N x N x N)
Output: matrix C (N x N x N)
initialization matrix C;
for $i$ = $0$ to $N$ do
    initialization matrix Temp2D;
    for $j$ = $0$ to $N$ do
        initialization matrix Temp1D;
        for $k$ = $0$ to $N$ do
            Temp1D $\leftarrow A_{ijk} + B_{ijk}$
        end
        Temp2D $\leftarrow Temp1D$
    end
    C $\leftarrow Temp2D$
end
```

Algorithm 2: Rank 3 Tensor Add

B. Multiplication

Similarly like addition, the multiplication algorithms are the same for both 2D and 3D multiplication. An entry for matrix C is the summation of the products between the corresponding row entries of matrix A and the corresponding column entries of matrix B. The difference between 2D multiplication and 3D multiplication is that before moving onto the next row or column entry of matrix C in 3D matrices, the next entry of matrix C goes into the next layer. For example, thinking of a

3D matrix as a cube in the 3D Cartesian plane, entry one can be located at [0,0,0]. The next entry will occur in [0,0,1], this indicates entries forming in the z-plane. Once all entries in the z-plane have been calculated, the next row or column entry is calculated. The pseudo code for 2D and 3D matrix multiplication can be seen below in Algorithm 3 and 4.

```
Input: matrix A (N x N), matrix B (N x N)
Output: matrix C (N x N)
initialization matrix C (N x N);
for i = 0 to N do
    for j = 0 to N do
        initialization temp to 0;
        for k = 0 to N do
            temp += A$_{ik} * B_{kj}$
        end
        C$_{ij} \leftarrow temp$
    end
end
```

Algorithm 3: Rank 2 Tensor Multiply

```
Input: matrix A (N x N x N), matrix B (N x N x N)
Output: matrix C (N x N x N)
initialization matrix C;
for i = 0 to N do
    initialization matrix Temp2D;
    for j = 0 to N do
        initialization matrix Temp1D;
        for k = 0 to N do
            initialization tempC to 0;
            for l = 0 to N do
                tempC += A$_{kjl} + B_{kli}$
            end
            Temp1D $\leftarrow tempC$
        end
        Temp2D $\leftarrow Temp1D$
    end
    C $\leftarrow Temp2D$
end
```

Algorithm 4: Rank 3 Tensor Multiply

IV. Analysis of the Algorithms

A. Algorithm Run-time

The efficiency of the algorithms is determined by analyzing the running time. The running time mainly depends on the size of the input, N. Using Big-Oh notation, the running times of the four algorithms are shown in table 1.

TABLE I
Running Time for Algorithms

Algorithm	Running Time
rank2TensorAdd()	$O(N^2)$
rank2TensorMult()	$O(N^2)$
rank3TensorAdd()	$O(N^3)$
rank3TensorMult()	$O(N^3)$

B. Data Structure Used

The data structures chosen to store the matrix elements are vectors, vectors are used as they can be returned from a function. This was particularly useful as using arrays would have resulted in memory being dynamically allocated. Arrays would have added a degree of complexity which was undesirable. Vectors alone are not the best solution as they require more memory than arrays. Furthermore, a better method of implementation would be the use of smart pointers. This would have faster run times without the added complexity of dynamically allocated memory.

V. Extending the Algorithm to Allow Matrix Multiplication in N-dimensions

A simple method can be used to extend the algorithm to N-dimensions. Using the function created for matrix multiplication for the previous dimension the next sequential dimension can be calculated. To multiply the matrices of a given dimension, the matrices should be split into equal layers of the previous dimension. For example while working with 12-D matrices the layers created needed to of the 11^{th} dimension. Using these layers along with the previous dimension's function for multiplication, the current dimensions multiplication can be obtained by looping through the new dimension. Every iteration in the new dimension will result in a layer of the previous dimension size which can be added to the current dimensions matrix C.

VI. Conclusion

Therefore, through the use of commonly used matrix addition and multiplication algorithms. Algorithms were created to calculate the Matrix Multiplication and Addition in both 2 and 3 dimensions. Pseudo code has been included for all the algorithms created. A method description has been discussed to extended Matrix Multiplication to any dimension.

References

[1] Ashu M. G. Solo, **Multidimensional Matrix Mathematics: Multidimensional Matrix Equality, Addition, Subtraction, and Multiplication, Part 2 of 6**, http://www.iaeng.org/publication/WCE2010/WCE2010$_{pp}1829-1833.pdf, date accessed 21/02/2019$.