ELEN4020: Data Intensive Computing Lab 1

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Elias Sepuru 1427726 Boikanyo Radiokana 1386807 Lloyd Patsika 1041888

I. INTRODUCTION

This report briefly describes the methods and algorithms used for solving the computations of a 2D and a 3D matrices of sizes $n \times n$ and $n \times n \times n$ respectively using C++ .

II. DESIGN & IMPLEMENTATION

The following subsections present the procedures and pseudocode followed in the addition of 2D matrices, addition of 3D matrices, multiplication of 2D matrices and multplication of 3D matrices.

A. Multiplication of 2D matrices

For the multiplication of 2D matrices, the following standard mathematical equation is followed:

$$c_{ij} = \sum_{k} a_{ik} \times b_{kj} \tag{1}$$

The function used for the multiplication of the 2D matrices is `rank2DTensorMult(A,B)` accepting two matrices A and B and returning a resultant matrix C as per equation 1. The following pseudocode represents the internal workings of the `rank2DTenosrMult(A,B)` function.

input : Two 2D matrices of sizes $n \times n$
output: A resultant C 2D matrix
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** n **do**
    for $k \leftarrow 1$ n **do**
      tempSum $\leftarrow$ tempSum + $A[i,k] \times B[j,k]$;
    end
    tempVector $\leftarrow$ tempSum;
  end
  C $\leftarrow$ tempVector;
end
return C

Algorithm 1: `rank2DTenosrMult(A,B)`: 2D Matrix Multplication

B. Addition of 2D matrices

For the addition of 2D matrices, the function `rank2DTensorAdd(n,A,B)` is used. The function accepts the matrices A and B and also accepts the desired size, n, inputed by the user. The function then returns a resultant C, which is an addition of A and B. Algorithm 2 below presents the pseudocode of the function.

input : size of matrix(n),Two 2D matrices of sizes $n \times n$
output: A resultant C 2D matrix
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** n **do**
    $C[i,j] \leftarrow A[i,k] + B[j,k]$;
  end
end
return C

Algorithm 2: `rank2DTenosrAdd(n,A,B)`: 2D Matrix Addition

C. Multiplication of 3D matrices

All the 3D computations adopt the 2D computations discussed in earlier sections. For 3D multiplication, the 3D$n \times n \times n$ matrices are broken down into n 2D matrices of sizes $n \times n$. From there they are passed into the function `rank2DTensorMult(A,B)` which returns $subC$ of size $n \times n$. The $n \times n$ $subC$ is then used to build an $n \times n \times n$ 3D matrix, C, layer by layer. All of this happens in `rank3DTensorMult(A,B)` function illustrated by agorithm 3.

input : Two 3D matrices of sizes $n \times n \times n$
output: A resultant 3D matrix, C
for $i \leftarrow 1$ **to** n **do**
  subC $\leftarrow$ rank2DTensorMult($A[i], B[i]$);
  C $[i] \leftarrow$ subC;
end
return C

Algorithm 3: `rank3DTenosrMult(A,B)`: 3D Matrix Multplication

D. Addition of 3D matrices

As mentioned earlier 2D computations are used to build up 3D computations. The addition of 3D matrices also adopts the function `rank2DTensorAdd(n,A,B)` on the break down of 3D matrices into 2D matrices. The function `rank3DTensorAdd(n,A,B)` is used and algorithm 4 illustrates the internal workings of the function.

input : Two 3D matrices of sizes $n \times n \times n$
output: A resultant 3D matrix, C
for $i \leftarrow 1$ **to** n **do**
    subC $\leftarrow$ rank2DTensorAdd($A[i], B[i]$);
    C $[i] \leftarrow$ subC;
end
return C

Algorithm 4: `rank3DTenosrAdd(n,A,B)`: 3D Matrix addition

E. Error Condition

The only possible error that can occur is when a user enters an invalid input for the size of the matrix. This error is conditioned by using the `assert` built-in function in C++ to notify and abort the program when the user inputs an invalid matrix size. An assumption that the size of the matrix must be greater than 1 is made.

III. CONCLUSION

The implementation of 2D and 3D matrix computations are succesfully executed. The computations are coded in C++. The 2D matrix computation code proved to be reusable in 3D computations, avoiding the DRY principle. The methodology used in the computation of the 3D matrices proved to be using sequential computing as each layer of the final matrix had to wait for another layer to be computed before it could be computed.