University of the Witwatersrand
School of Electrical & Information Engineering

ELEN4020A
Data Intensive Computing for Data Science
Laboratory 2 Report

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

Due Date: 20 March 2019

***Abstract*—This report discusses the laboratory work done on the transposition of 2-dimensional square matrices. Six algorithms that compute the transpose of an array are developed. One algorithm is the basic transposition algorithm and the remaining algorithms use parallel programming using two libraries; namely PThread and OpenMP. A main program that dynamically generates the elements of the input arrays is developed. The elements are randomly generated between [0, 20000]. The corresponding pseudo code for ... out of the six algorithms is included. The algorithms are tested with matrices of sizes 128 $\times$ 128, 1024 $\times$ 1024, 2048 $\times$ 2048 and 4096 $\times$ 4096. It is found that the Op nMP Naiveealgorithm is the fastest.**

I. Introduction

Matrix transposition is a relatively simple process that can be achieved using the basic transposition algorithm. The issue arises when there is a need to transpose large matrices. Parallel programming solves this issue by transposing different sections of the matrix at the same time. This report discusses the development of six algorithms for matrix transposition. Included are the lab requirements and the pseudo code for the algorithms. These can be found in sections II and III respectively. The time it takes the algorithms to transpose matrices of different sizes is in section IV. Lastly a method for transposing matrices that are not stored on memory is 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 six separate procedures which are to handle the transposition of 2-D matrices. The input matrices are to be sized 128×128, 1024×1024, 2048×2048 and then 4096×4096. These procedures are the:

- Basic Algorithm
- PThread Diagonal-Threading Algorithm
- PThread Block Oriented-Threading Algorithm
- OpenMP Naive-Threading Algorithm
- OpenMP Diagonal-Threading Algorithm
- OpenMP Block Oriented-Threading

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

Five out of the six transposition algorithms make use of parallel programming.

A. Basic Algorithm

This is a basic transposition algorithm which makes use of a nested for loop. In this algorithm corresponding row and column entries are simply swapped. The psuedo code for the basic algorithm can be seen in Algorithm 1.

```
Input: matrix A (row x colm), integer N
Output: matrix A (row x colm), integer N
for i = 0 to row do
    for j = 0 to colm do
        initialization: temp;
        temp ← A_ij
        A_ij ← A_ji
        A_ji ← temp
    end
end
```

Algorithm 1: Basic Transposition Algorithm

B. Pthreads: Diagonal

The PThread diagonal algorithm is descried by the pseudo code below. The PThread algorithm swaps the vertical column and horizontal row of each diagonal element. The code cycles through every diagonal element and creating a separate thread for each one.

```
Input: matrix A (row x colm)
Output: matrix B (row x colm)
initialization: matrix B (row x colm);
initialization: matrix Temp1 (row);
initialization: matrix Temp2 (colm);
pthreadt newthread[rol x colm];
for i = 0 to row do
    for j = 0 to colm do
        if i == j then
            for x = i to row do
                pthreadcreate(newthread[i]);
                pthreadjoin(newthread[i], NULL);
            end
        end
    end
end
```

Algorithm 2: PThread Diagonal-Threading Algorithm

C. Pthreads: Blocked

The Block Pthreading transposition method makes use of smaller memory blocks in-order to shift the values around the original matrix. A struct is created that holds four elements. The elements contained in the struct represent a single block. Therefore, an array of struct blocks are created to hold all the element of the array. The size of this array is the same as the size of the columns and rows. The elements in the blocks are first transposed and then swapped into the correct position as described by the lab brief.

D. OpenMP: Naive

The OpenMP Naive-Threaded algorithm is a simple transposition algorithm which makes use of nested for loops to swap A[i,j] with A[j,i]. This is similar to the basic algorithm, however *#pragma omp parallel* is inserted to make the algorithm parallel. The Algorithm can be seen in Algorithm 1.

E. OpenMP: Diagonal

The OpenMP Diagonal-Threading algorithm is similar to the PThread diagonal algorithm. For each diagonal position in the matrix, the corresponding row and column entries are swapped. This algorithm also makes use of *#pragma omp parallel* to create threads that transpose different sections of the matrix in parallel. The algorithm can be seen below in Algorithm 3.

Input: matrix A (row x colm)
Output: matrix A (row x colm)
#pragma omp parallel;
initialization: matrix Temp1 (row);
initialization: matrix Temp2 (colm);
for *i = 0 to row* **do**
  for *j = 0 to colm* **do**
    if *i == j* **then**
      for *x = i to row* **do**
        Temp1$_x \leftarrow A_{ix}$
        Temp2$_x \leftarrow A_{xj}$
      end
      for *y = i to row* **do**
        A$_{iy} \leftarrow Temp2_y$
        A$_{yj} \leftarrow Temp1_y$
      end
    end
  end
end

Algorithm 3: OpenMP Diagonal-Threading Algorithm

F. OpenMP: Blocked

This algorithm uses double transposition. The matrix to be transposed is first divided into smaller square matrices, called blocks. The number of blocks corresponds to the number of threads and as such can be user defined. For the first round of transpositions the entries in the blocks are transposed using the basic transposition algorithm. After this, the blocks themselves are then transposed completing the second round of transposition.

IV. Analysis of the Algorithms

The efficiency of the algorithms is determined by analyzing the running time. The runningti mes shown in table 1 a.eri nesondcs

TABLE I
RUNNING TIME FOR ALGORITHMS

N_0	Basic	PThd (D)	PThd (B)	OMP (N)	OMP (D)	OMP (B)
128	1.203	0.074	1.116	0.202	0.338	1.66
1024	2.315	2.168	3.856	0.402	1.746	3.307
2048	4.387	5.401	7.701	0.607	2.345	6.463
4096	9.479	11.682	12.980	0.987	3.755	9.344

V. Adjusting The Algorithm if The Matrices Were Not Stored in Memory

A simple change to the code structure could be made if the matrices were not stored in memory, this could be done by using an external text file. The code could simply generate the numbers and write them onto an external text file. The same text file would be used to read in the entries for the desired transposed matrix, a different text file or the same text file could be used to write the output of the transposed matrix.

VI. Conclusion

Therefore, through the use of multiple matrix transposition algorithms the lab was successfully completed. The algorithms were created to calculate matrix transposition of multiple large sized matrices . Pseudo code has been included for all the algorithms created. Further discussion were made on the runtime and effectiveness of these algorithms.