ELEN4020: Data Intensive Computing Lab 2

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 document provides a brief explanation of the algorithms used to transpose 2D large matrices using shared memory programming libraries, Pthread and OpenMP. The parallel programming libraries are compiled in the Linux environment using C. The performance of the different algorithms with and without the parallel programming libraries are compared and analysed. The sizes of the matrices that are transposed are: 128, 1024, 2048 and 4096 respectively. The number of threads for both Pthreads and OpenMP are set to 8 for all the algorithms.

II. NAIVE TRANSPOSITION

A. Basic

The algorithm involves iterating the whole 2D matrix using two nested for loops for the row and column respectively. The element at index [i,j] is swapped with the element at index [j,i] using a temporary variable. The pseudo code 4 below shows the algorithm used in the `NaiveTransposeBasic(twoD)` function.

input: 2D matrix of size $n \times n$
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** i **do**
    temp $\leftarrow A[i,j]$;
    $A[i,j] \leftarrow A[j,i]$;
    $A[j,i] \leftarrow$ temp;
  end
end

Algorithm 1: `NaiveTransposeBasic(twoD)`

B. OpenMP

The number of threads set for Naive threading is set at 8. The varaibles i and j, indices of rows and columns are defined inside the pragma statement which sets them to be private by default. The directives used for the parallelism of the two nested for loops are:

- **`#pragma omp parallel`**
- **`#pragama omp for nowait`**

The use of the `nowait` clause allows threads that have completed the tasks to carry on with the next code in the parallel region. The second directive distributes the iterations of the for loop to the threads for execution. The algorithm 9 below shows the pseudo code implemented in the function.

input: 2D matrix of size $n \times n$
`#pragma omp parallel`
for $i \leftarrow 0$ **to** $n-1$ **do**
  `#pragma omp for nowait`
  for $j \leftarrow 0$ **to** i **do**
    temp $\leftarrow A[i,j]$;
    $A[i,j] \leftarrow A[j,i]$;
    $A[j,i] \leftarrow$ temp;
  end
end

Algorithm 2: `NaiveTransposeBasic(twoD)`

III. DIAGONAL TRANSPOSITION

A. Basic

The algorithm involves iterating the 2D matrix along the main diagonal of the matrix. At each diagonal, the matrix is transposed by swapping the elements in the corresponding row with the elements in the corresponding column. The element at index [i,j] is swapped with the element at index [j,i].

input: 2D matrix of size $n \times n$
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow i+1$ **to** $n-1$ **do**
    temp $\leftarrow A[i,j]$;
    $A[i,j] \leftarrow A[j,i]$;
    $A[j,i] \leftarrow$ temp;
  end
end

Algorithm 3: `DiagonalTransposeBasic(twoD)`

B. OpenMP

The same methodology used in the Naive-OpenMP threading is used because the structure of the code is the same, the only difference is the way the matrix is accessed which can be seen in the pseudo code above. The two directives used are:

- **`#pragma omp parallel`**
- **`#pragama omp for nowait`**

The pseudo code below shows the use of OpenMP threads in the diagonal threading algorithm.

```
input: 2D matrix of size n × n
#pragma omp parallel
for i ← 0 to n − 1 do
    #pragma omp for nowait
    for j ← i + 1 to n − 1 do
        temp ←A[i, j];
        A[i, j]←A[j, i];
        A[j, i]←temp;
    end
end
```

Algorithm 4: `DiagonalTransposeBasic(twoD)`

C. Pthreads

The Diagonal Transpose using Pthreads algorithm is built on top of the basic diagonal transpose method. This means that the swapping along the diagonals is the same, however the code is now executed using 8 parallel threads. Each thread is initially assigned to a diagonal, and this assignment is dependent on the size of the matrix. Since the sizes for this lab's matrices are all greater than 8, each thread does get assigned an initial diagonal to perform a diagonal transpose on. The need for the threads to be re-assigned new diagonals as well as make sure the threads are always in sync and do not edit an already transposed diagonal resulted in: a `While-Loop` that continues until a `break` due to the next possible thread being the second last diagonal, `N-1`, as well as `Pthread_Mutex_Lock` and `Pthread_Mutex_Unlock` being used to keep the threads in sync, as well as assign a new diagonal to a thread. The pseudo code can be seen in Algorithm 5.

```
input: A Struct with 2D matrix of size n × n,
       Diagonal, Matrix Size
while true do
    for i ← 0 to n − 1 do
        for j ← i + 1 to n − 1 do
            temp ←A[i, j];
            A[i, j]←A[j, i];
            A[j, i]←temp;
        end
    end
    Pthread_Mutex_Lock
    if nextDiagonal < SIZE-1 then
        currentDiagonal = nextDiagonal
        ++nextDiagonal
    end
    else
        currentDiagonal = SIZE-1
    end
    Pthread_Mutex_Unlock
    if currentDiagonal == SIZE-1 then
        BREAK
    end
end
```

Algorithm 5: `DiagonalTransposePThread(*agms)`

IV. Blocked Transposition

A. Basic

Block Transposition of a matrix involves splitting the matrix in to equal sub-matrices, each sub-matrix is then transposed. After the transposition of each sub-matrix, the sub-matrices are 'seen' as elements of the original/bigger matrix. The original matrix is then transposed with elements being the sub-matrices. Algorithm 6 represent the algorithm that was used to transpose each sub-matrix.

To transpose the blocks the function `blockTranspose(A)` was used. Algorithm 7 presents the pseudocode for the `blockTranspose(A)` function.

The `blockSwap` as seen in Algorithm 7 takes in the positions of the blocks where the swaps take place together with a pointer to the matrix.

```
input: A 2D matrix "A" of size n × n
otherCol ←0
otherRow ←0
for i ← 0 to n − 1 do
    for j ← 0 to n − 1 do
        for k ← 0 to BLOCK_SIZE do
            for l ← 0 to l < k do
                if i! = j then
                    otherRow ← (i + l) otherCol
                      ← (j + k)
                end
                else
                    otherRow ← (l + j) otherCol
                      ← (k + i)
                end
                swap (A[k + i][l + j],
                  A[otherRow][otherCol])
            end
        end
    end
end
```

Algorithm 6: `blockElementTranspose(A)`

B. Pthreads

The Pthreads block threading algorithm was built on top of the basic one. The functions depicted by Algorithm 6 & 7 were altered in way that Pthreads function would work on them. Algorithm 8 illustrates how the threads were created for block element transposition.

For block transposition a similar approach was followed. Algorithm 9 illustrates.

input: 2D matrix "A" of size $n \times n$
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** $j < i$ **do**
    blockSwap (i, j, A)
  end
end

Algorithm 7: `blockTranspose(A)`

input: A 2D matrix "A" of size $n \times n$
while *1* **do**
  otherCol $\leftarrow$0
  otherRow $\leftarrow$0
  for $i \leftarrow$ blockThread$->row$ **to** blockThread$->row + BLOCK_SIZE$ **do**
    for $j \leftarrow 0$ **to** $n-1$ **do**
      for $k \leftarrow 0$ **to** $BLOCK\_SIZE$ **do**
        for $l \leftarrow 0$ **to** $l < k$ **do**
          if $i! = j$ **then**
            otherRow $\leftarrow (i+l)$ otherCol $\leftarrow (j+k)$
          end
          else
            otherRow $\leftarrow (l+j)$ otherCol $\leftarrow (k+i)$
          end
          swap ((blockThread$->arr + (k+i) * SIZE + (l+j)$), (blockThread$->arr +$ (otherRow) $* SIZE +$ (otherCol))
        end
      end
    end
  end
  if $NEXT\_ROW\_E >= SIZE - 1$ **then**
    break
  end
  $pthread\_mutex\_lock(\&stillBusy)$;
  blockThread$->row \leftarrow NEXT_ROW_E$;
  $NEXT\_ROW\_E+ = BLOCK\_SIZE$
  $pthread\_mutex\_unlock(\&stillBusy)$
end

Algorithm 8: `PthreadblockElementTranspose(A)`

C. OpenMP

The functions `blockElementTranspose()` and `blockTranspose()` are the only functions in the build up of block threading were OpenMP is implemented. The directive used in `blockElementTranspose()` is a one line parallel construct with the for, private and shared clause.

- **`pragma omp parallel for private(otherRow, otherCol) shared(twoD)`**

This means that the variables in the private function are not shared between the threads,each thread has its own local

input: 2D matrix "A" of size $n \times n$
while *1* **do**
  for $i \leftarrow -> row$ **to** $-> row + BLOCK\_SIZE$ **do**
    for $j \leftarrow 0$ **to** $j < i$ **do**
      blockSwap (i, j, A)
    end
  end
  if $NEXT\_ROW\_B >= SIZE - 1$ **then**
  end
  $pthread\_mutex\_lock(\&stillBusy)$;
  $-> row \leftarrow NEXT\_ROW\_$;
  $NEXT\_ROW\_B+ = BLOCK\_SIZE$
  $pthread\_mutex\_unlock(\&stillBusy)$
end

Algorithm 9: `blockTranspose(A)`

copy of the variable. However the twoD array in the shared function is shared between the threads.This means that there is only one instance of the array and multiple threads can access the twoD array at the same time. The directives used in the `blockTranspose()` function is the same as the Diagonal Transposition for OpenMP since the methodology is the same. Algorithm 10 shows the pseudo code of the OpenMP Block Threading.

input: 2D matrix of size $n \times n$
`#pragma omp parallel`
for $i \leftarrow 0$ **to** $i + BLOCK\_SIZE$ **do**
  `#pragma omp for nowait`
  for $j \leftarrow 0$ **to** $j + BLOCK\_SIZE$ **do**
    `blockSwap(i,j,twoD[0])`
  end
end

Algorithm 10: `blockTransposeBasic(twoD)`

D. Pthreads

V. Timer

The function `timer(twoD, type_transpose)` is a void function that accepts any of the transposing functions as an argument together with a string which is a name of the method used. It uses the $gettimeofday()$ function from the sys/time.h preprocessor directive. The timer starts counting just before the transposing function that is passed to the timer function is called. After the transposing function executes the mathematical computations the timer stops counting. The duration of the transposing function is calculated as seen in Algorithm 11 below .The units of the duration is presented in milliseconds.

VI. Results & Analysis

The algorithms were run 4 times(see Appendix A) on the same machine at the same conditions to ensure fairness. The average of the 5 readings for each method and size were

input: 2D matrix of size $n \times n$
`gettimeofday` (& start, $NULL$)
`f` (twoD)
`gettimeofday` (& end, $NULL$)
duration ←((end-start)+1e6*(end-start))*1000
display duration

Algorithm 11: `DiagonalTransposeBasic(twoD)`

calculated and recorded in table below.

TABLE I
PERFOMANCE OF DIFFERENT SIZES WITH DIFFERET METHODS OF THREADING

Matrix sizes nxn	Basic	Pthreads		OpenMP		
		Diagonal	Blocked	Naive	Diagonal	Blocked
128	0.22425	2.195	11.09	1.066	0.639	0.975
1024	11.311	3.123	4.321	4.201	3.329	4.849
2048	51.452	35.67	17.324	13.925	16.092	13.88
4096	244.9605	129.8487	67.364	76.67	119.1075	57.079

From the table above it can be seen the all the basic approaches perform relatively well from sizes 128-2048, however as the sizes increases the threading algorithms prove to be more efficient.

VII. CONCLUSION

From the results gathered in the table it can be seen that Block Threading is faster in all of the implementations, with OpenMP taking first place in the implementations.

APPENDIX

```
--------------------------------128----------------------------

Basic Naive-Transposition      t = 0.29 milliseconds
OpenMP Naive-Threading         t = 0.986 milliseconds
Basic Diagonal-Transposition   t = 0.296 milliseconds
OpenMP Diagonal-Threading      t = 0.358 milliseconds
Pthread Diagonal-Threading     t = 1.422 milliseconds
Basic Block-Transposition      t = 0.558 milliseconds
OpenMP Block-Threading         t = 1.666 milliseconds
Pthread Block-Threading        t = 3.514 milliseconds

-----------------------------------------------------------------


--------------------------------1024----------------------------

Basic Naive-Transposition      t = 9.389 milliseconds
OpenMP Naive-Threading         t = 2.638 milliseconds
Basic Diagonal-Transposition   t = 11.581 milliseconds
OpenMP Diagonal-Threading      t = 2.914 milliseconds
Pthread Diagonal-Threading     t = 5.297 milliseconds
Basic Block-Transposition      t = 11.735 milliseconds
OpenMP Block-Threading         t = 3.922 milliseconds
Pthread Block-Threading        t = 8.832 milliseconds

-----------------------------------------------------------------


--------------------------------2048----------------------------

Basic Naive-Transposition      t = 54.06 milliseconds
OpenMP Naive-Threading         t = 12.184 milliseconds
Basic Diagonal-Transposition   t = 62.283 milliseconds
OpenMP Diagonal-Threading      t = 12.467 milliseconds
Pthread Diagonal-Threading     t = 34.605 milliseconds
Basic Block-Transposition      t = 42.521 milliseconds
OpenMP Block-Threading         t = 16.834 milliseconds
Pthread Block-Threading        t = 21.412 milliseconds

-----------------------------------------------------------------


--------------------------------4096----------------------------

Basic Naive-Transposition      t = 254.408 milliseconds
OpenMP Naive-Threading         t = 74.547 milliseconds
Basic Diagonal-Transposition   t = 244.837 milliseconds
OpenMP Diagonal-Threading      t = 78.434 milliseconds
Pthread Diagonal-Threading     t = 133.718 milliseconds
Basic Block-Transposition      t = 138.02 milliseconds
OpenMP Block-Threading         t = 63.659 milliseconds
Pthread Block-Threading        t = 61.596 milliseconds
```

Fig. 1. Caption

```
--------------------------------128----------------------------

Basic Naive-Transposition      t = 0.234 milliseconds
OpenMP Naive-Threading         t = 1.294 milliseconds
Basic Diagonal-Transposition   t = 0.317 milliseconds
OpenMP Diagonal-Threading      t = 1.296 milliseconds
Pthread Diagonal-Threading     t = 2.072 milliseconds
Basic Block-Transposition      t = 0.609 milliseconds
OpenMP Block-Threading         t = 0.775 milliseconds
Pthread Block-Threading        t = 2.271 milliseconds

-----------------------------------------------------------------



--------------------------------1024----------------------------

Basic Naive-Transposition      t = 10.625 milliseconds
OpenMP Naive-Threading         t = 3.992 milliseconds
Basic Diagonal-Transposition   t = 12.711 milliseconds
OpenMP Diagonal-Threading      t = 3.887 milliseconds
Pthread Diagonal-Threading     t = 5.818 milliseconds
Basic Block-Transposition      t = 11.036 milliseconds
OpenMP Block-Threading         t = 5.414 milliseconds
Pthread Block-Threading        t = 4.777 milliseconds

-----------------------------------------------------------------



--------------------------------2048----------------------------

Basic Naive-Transposition      t = 47.066 milliseconds
OpenMP Naive-Threading         t = 12.922 milliseconds
Basic Diagonal-Transposition   t = 56.173 milliseconds
OpenMP Diagonal-Threading      t = 13.977 milliseconds
Pthread Diagonal-Threading     t = 29.219 milliseconds
Basic Block-Transposition      t = 35.821 milliseconds
OpenMP Block-Threading         t = 14.428 milliseconds
Pthread Block-Threading        t = 16.409 milliseconds

-----------------------------------------------------------------



--------------------------------4096----------------------------

Basic Naive-Transposition      t = 243.186 milliseconds
OpenMP Naive-Threading         t = 69.095 milliseconds
Basic Diagonal-Transposition   t = 238.053 milliseconds
OpenMP Diagonal-Threading      t = 67.012 milliseconds
Pthread Diagonal-Threading     t = 118.526 milliseconds
Basic Block-Transposition      t = 138.835 milliseconds
OpenMP Block-Threading         t = 51.741 milliseconds
Pthread Block-Threading        t = 68.609 milliseconds
```

Fig. 2. Caption

```
---------------------------------128-----------------------------

Basic Naive-Transposition      t = 0.094 milliseconds
OpenMP Naive-Threading         t = 0.579 milliseconds
Basic Diagonal-Transposition   t = 0.097 milliseconds
OpenMP Diagonal-Threading      t = 0.137 milliseconds
Pthread Diagonal-Threading     t = 0.96 milliseconds
Basic Block-Transposition      t = 0.228 milliseconds
OpenMP Block-Threading         t = 1.025 milliseconds
Pthread Block-Threading        t = 1.191 milliseconds

------------------------------------------------------------------


---------------------------------1024----------------------------

Basic Naive-Transposition      t = 9.838 milliseconds
OpenMP Naive-Threading         t = 2.103 milliseconds
Basic Diagonal-Transposition   t = 12.839 milliseconds
OpenMP Diagonal-Threading      t = 2.513 milliseconds
Pthread Diagonal-Threading     t = 6.551 milliseconds
Basic Block-Transposition      t = 8.751 milliseconds
OpenMP Block-Threading         t = 4.334 milliseconds
Pthread Block-Threading        t = 7.481 milliseconds

------------------------------------------------------------------


---------------------------------2048----------------------------

Basic Naive-Transposition      t = 46.908 milliseconds
OpenMP Naive-Threading         t = 14.379 milliseconds
Basic Diagonal-Transposition   t = 59.051 milliseconds
OpenMP Diagonal-Threading      t = 15.862 milliseconds
Pthread Diagonal-Threading     t = 31.474 milliseconds
Basic Block-Transposition      t = 36.799 milliseconds
OpenMP Block-Threading         t = 14.627 milliseconds
Pthread Block-Threading        t = 16.955 milliseconds

------------------------------------------------------------------


---------------------------------4096----------------------------

Basic Naive-Transposition      t = 237.515 milliseconds
OpenMP Naive-Threading         t = 79.526 milliseconds
Basic Diagonal-Transposition   t = 236.828 milliseconds
OpenMP Diagonal-Threading      t = 104.999 milliseconds
Pthread Diagonal-Threading     t = 128.243 milliseconds
Basic Block-Transposition      t = 142.653 milliseconds
OpenMP Block-Threading         t = 52.389 milliseconds
Pthread Block-Threading        t = 67.089 milliseconds
```

Fig. 3. Caption

```
---------------------------------128-----------------------------

Basic Naive-Transposition      t = 0.279 milliseconds
OpenMP Naive-Threading         t = 1.406 milliseconds
Basic Diagonal-Transposition   t = 0.297 milliseconds
OpenMP Diagonal-Threading      t = 0.669 milliseconds
Pthread Diagonal-Threading     t = 3.511 milliseconds
Basic Block-Transposition      t = 0.687 milliseconds
OpenMP Block-Threading         t = 0.934 milliseconds
Pthread Block-Threading        t = 11.09 milliseconds

------------------------------------------------------------------


---------------------------------1024-----------------------------

Basic Naive-Transposition      t = 10.457 milliseconds
OpenMP Naive-Threading         t = 4.201 milliseconds
Basic Diagonal-Transposition   t = 11.916 milliseconds
OpenMP Diagonal-Threading      t = 3.239 milliseconds
Pthread Diagonal-Threading     t = 5.479 milliseconds
Basic Block-Transposition      t = 10.999 milliseconds
OpenMP Block-Threading         t = 4.849 milliseconds
Pthread Block-Threading        t = 4.258 milliseconds

------------------------------------------------------------------


---------------------------------2048-----------------------------

Basic Naive-Transposition      t = 51.452 milliseconds
OpenMP Naive-Threading         t = 13.925 milliseconds
Basic Diagonal-Transposition   t = 57.732 milliseconds
OpenMP Diagonal-Threading      t = 16.093 milliseconds
Pthread Diagonal-Threading     t = 35.67 milliseconds
Basic Block-Transposition      t = 35.701 milliseconds
OpenMP Block-Threading         t = 13.888 milliseconds
Pthread Block-Threading        t = 17.324 milliseconds

------------------------------------------------------------------


---------------------------------4096-----------------------------

Basic Naive-Transposition      t = 244.733 milliseconds
OpenMP Naive-Threading         t = 83.517 milliseconds
Basic Diagonal-Transposition   t = 251.466 milliseconds
OpenMP Diagonal-Threading      t = 112.997 milliseconds
Pthread Diagonal-Threading     t = 138.908 milliseconds
Basic Block-Transposition      t = 146.72 milliseconds
OpenMP Block-Threading         t = 60.511 milliseconds
Pthread Block-Threading        t = 72.162 milliseconds
```

Fig. 4. Caption