



## CS434: Introduction to Parallel and Distributed Computing Laboratory Exercise 2 Parallel Programming with PThread and OpenMP

To Be Completed By: 11:59Hrs, February 17, 2020

### Outcome

This is a group assignment. The main purpose of this laboratory exercise is to learn to use shared memory programming libraries, Pthread and OpenMP, for simple scientific applications of manipulating very large matrices that are maintained in memory. The main outcome will be:

- i.) learning to use the Pthread and OpenMP parallel programming libraries.
- ii.) learning to compile and run parallel shared memory programs in Linux environment.
- iii.) learning to evaluate the performance of your parallel programs by timing.

### Problem Description

You are expected to write a program that computes the transpose of a **square matrix**  $A[N_0][N_1]$  in-place for  $N_0 = N_1$  and for  $N_0 = 128, 1024, 2048$  and  $4096$ . Computing the transpose in place requires that you do not create a second  $N_0 \times N_1$  matrix, say  $A^T[N_0][N_1]$ , and copy elements of the original matrix  $A$ , into  $A^T$  to contain the transposed elements of the original matrix. The elements of the original matrix should simply be reshuffled, using a small temporal storage space much much less than  $N_0 \times N_1$ , to generate the transposed elements in the original matrix. For example if the original matrix is a  $4 \times 4$  matrix  $A[4][4]$  as shown below,

$$A[4][4] = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

the transposed matrix should be

$$A^T[4][4] = \begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix}$$

The number of threads in both the PThread program and the OpenMP program should be chosen appropriately according to the suggested algorithm explained below. For each of the values of  $N_0$  and number of threads used, evaluate the time it takes to perform the transposition of the matrix, not including the time to populate the original matrix. Elements of the matrix  $A[N_0][N_1]$  should be a 4-byte integer, generated as random integers between 0 and  $N_0$ . We will consider for the use of PThreads:

1. A Diagonal-Threading Algorithm

## 2. A Block-Oriented Transposition.

For OpenMP:

1. A Naive-OpenMP Algorithm
2. A Block-Oriented OpenMP Transposition.

### OpenMP Naive-Threaded Algorithm

A simple non-threaded approach, or serial algorithm, is to transpose the given matrix with a simple nested loop by swapping the elements  $A\langle i, j \rangle$  with  $A\langle j, i \rangle$ . Call this method the Basic-Algorithm.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Figure 1: Example Matrix

1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63
8	16	24	32	40	48	56	64

Figure 2: Transposed Matrix

A naive OpenMP threading will instrument the serial algorithm by inserting `#PRAGMA` before for loops, etc., to automatically parallelize the code if compiled as an OpenMP application.

### Diagonal-Threading

In this algorithm, threads are generated for each diagonal elements. For each diagonal position, the corresponding row elements to the right and the column elements below are interchanged by the threads assigned to the position (See Figure 3).

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Figure 3: Diagonal-Threading

## Block-Oriented Transposition

The Block-Oriented-Threading can be perceived as 2-levels of transpositions. The matrix is chunked into blocks. The elements within each block are first transposed and then the blocks are then transposed to generate the overall transposed matrix. See the illustration of the algorithm below in Figure 4.

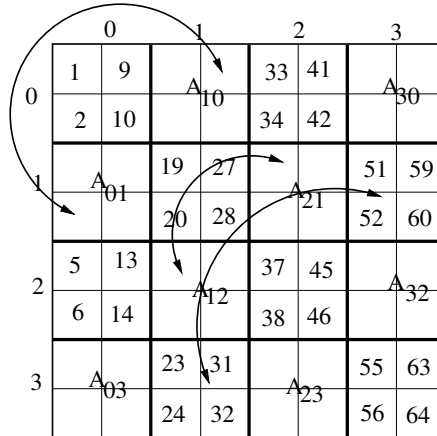


Figure 4: Block-Oriented Transposition

## Work Schedule

The work will involve:

1. developing and implementing a multi-threaded algorithm for computing the transpose of a matrix in place in PThread and OpenMP. Your algorithm should be as efficient and also scalable as much as possible.
2. generating a comparative table of the performance of your algorithm as in the sample table below.

$N_0 = N_1$	Basic	Pthreads		OpenMP	
		Diagonal	Blocked	Naive	Blocked
128					
1024					
2048					
4096					

## The Deliverable

- Submit your codes, for marking in your group's initialised repository on GitHub.
- Provide a short description of your solution to this problem and possible the pseudo-codes of your routines.
- Write a short set of instructions on how to access your GitHub repository and send this to CANVAS. This should be submitted by only the lead member of your group.

## References

- Lecture Notes on PThread and OpenMP
- <https://computing.llnl.gov/tutorials/pthreads/>
- <https://computing.llnl.gov/tutorials/openMP/>