ELEN4020A: Data Intensive Computing
Course Project (Simplified Alternative): Matrix Transposition of Big-Data

To Be Completed By Friday May 13, 2019 at 17:55pm

Preamble

You are expected to work in your normal groups of 3. Each member is expected to contribute equal amount of work as the other members of the group. Same marks will be assigned to every member. You are expected to work with a good understanding of the background knowledge required; namely:

- OpenMP: Shared Memory
- MapReduce Programming Model in Shared Memory
- Message Passing Interface (MPI): One sided communication with put()/get();
- Parallel I/O: MPI-IO, Filetype creation, collective IO, etc.
- PGAS model of parallel computation: more specifically UPC/UPC++.

Note that due to time constraint, a particular programming model may not be covered in detail in class. However a high-level coverage will still be done. In this case you may, for the purposes of the project, have to read about the basics (Pointers to tutorial will be provided) and discuss the details with the instructor. Further explanation, if needed, can be given during office hours. The general requirements of the project involve performance comparison of a matrix transposition problem using two programming models; a straight forward MPI parallel programming method versus any *one* of the following:

i. One-Sided Communication with Derived Data Types

ii. UPC/UPC++

The choice of the other programming method is left to the group to decide, i.e., slightly different algorithmic approach to solving the same problem. The original Global_Array approach gives errors during testing with included libraries of blas, scalapack, etc.

Detailed Problem Description

The problem definition is simply to develop and implement a parallel matrix transposition algorithm for very large data-sets that should be generated by a group of processors as sub-matrices but together form a large matrix say *matrixFile_N*. The overall matrix should form a large square $N \times N$ matrix $A[N][N]$. The overall matrix can be written by the master process of rank 0. Suppose processor 0 is to write this into a file or a simple output, the first value in the file of *matrixFile_N*, specifies the value of N. The rest of the values indicate the elements $a_{i,j} = A[i][j]$ which are short integers. The values should be separated by spaces or commas as you choose. The ordering can simply be in row-major order.

Input Data, *matrixFile_N*

The elements of the global input matrix A[N][N] can be perceived as being in a row-major order if these were to be written out as shown in Tables 1 and 2. Unlike the previous problem where collective I/O was required, each sub-matrix can generated elements of its original sub-matrix as small random integers between 0 and 99.

Table 1: Matrix A[8][8]

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$
$a_{4,0}$	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	$a_{4,7}$
$a_{5,0}$	$a_{5,1}$	$a_{5,2}$	$a_{5,3}$	$a_{5,4}$	$a_{5,5}$	$a_{5,6}$	$a_{5,7}$
$a_{6,0}$	$a_{6,1}$	$a_{6,2}$	$a_{6,3}$	$a_{6,4}$	$a_{6,5}$	$a_{6,6}$	$a_{6,7}$
$a_{7,0}$	$a_{7,1}$	$a_{7,2}$	$a_{7,3}$	$a_{7,4}$	$a_{7,5}$	$a_{7,6}$	$a_{7,7}$

Table 2: Data *matrixFile_N*.

8 $a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$ $a_{0,4}$ $a_{0,5}$ $a_{0,6}$ $a_{0,7}$ $a_{1,0}$ $a_{1,1}$ $a_{1,2}$ ... $a_{7,6}$ $a_{7,7}$

Computing the Transpose

In the matrix of Table 1, the lines shown are boundaries of chunking the array into chunks of 2×2 elements. One can now consider the chunks as being assigned to 16 processes $\{P_0, P_1, P_2, \ldots, P_15$ } in exactly row major order. For example if the chunks are considered as a 4×4 matrix C[4][4], the sub matrix C[0][2] is assigned to processor P_2, and C[2][3] is assigned to processor P_{11}.

After the transposition the elements of sub matrix C[2][1], previous stored by processor P_9y should now be assigned to processor P_7 as sub-matrix C[1][2] except that the elements will now be transposed.

The output Data, *matrixFile_N*

The elements of the transposed matrix A[N][N] are still stored in a row-major order in the global array *matrixFile_N* as shown in Tables 3 and 4.

Table 3: Matrix A[8][8]

$a_{0,0}$	$a_{1,0}$	$a_{2,0}$	$a_{3,0}$	$a_{4,0}$	$a_{5,0}$	$a_{6,0}$	$a_{7,0}$
$a_{0,1}$	$a_{1,1}$	$a_{2,1}$	$a_{3,1}$	$a_{4,1}$	$a_{5,1}$	$a_{6,1}$	$a_{7,1}$
$a_{0,2}$	$a_{1,2}$	$a_{2,2}$	$a_{3,2}$	$a_{4,2}$	$a_{5,2}$	$a_{6,2}$	$a_{7,2}$
$a_{0,3}$	$a_{1,3}$	$a_{2,3}$	$a_{3,3}$	$a_{4,3}$	$a_{5,3}$	$a_{6,3}$	$a_{7,3}$
$a_{0,4}$	$a_{1,4}$	$a_{2,4}$	$a_{3,4}$	$a_{4,4}$	$a_{5,4}$	$a_{6,4}$	$a_{7,4}$
$a_{0,5}$	$a_{1,5}$	$a_{2,5}$	$a_{3,5}$	$a_{4,5}$	$a_{5,5}$	$a_{6,5}$	$a_{7,5}$
$a_{0,6}$	$a_{1,6}$	$a_{2,6}$	$a_{3,6}$	$a_{4,6}$	$a_{5,6}$	$a_{6,6}$	$a_{7,6}$
$a_{0,7}$	$a_{1,7}$	$a_{2,7}$	$a_{3,7}$	$a_{4,7}$	$a_{5,7}$	$a_{6,7}$	$a_{7,7}$

Table 4: Data *matrixFile_N*.

8 $a_{0,0}$ $a_{1,0}$ $a_{2,0}$ $a_{3,0}$ $a_{4,0}$ $a_{5,0}$ $a_{6,0}$ $a_{7,0}$ $a_{0,1}$ $a_{1,1}$ $a_{2,1}$ ... $a_{6,7}$ $a_{7,7}$

The Details of the Algorithms

The work requires that you conduct two algorithms and plot the performance the wall-time to compute the transpose. This includes clocking the time to read and write the data. Your output file name should be different from your output file.

Algorithm-1: This requires using one-sided-communication, i.e., setting up windows by each processor, using MPI_Get() and MPI_Get(), etc., to perform the transposition of the matrix.

Algorithm-2 This requires using only *one* of the PGAS model of programming after reading the arrays from the input files:

- UPC/UPC++ approach
- Global_Array approach

The algorithms are to be run for array sizes of $N = 2^n$ for $n = \{3, 4, 5, 6, 7\}$. You should test these for $P = \{16, 32, 64\}$ processors. The reading and writing of elements of the matrix into structures held by the processors should use **collective-I/O**.

Input Data

You may generate input files from simple random number generation of short integer numbers and store the array elements in in row-major order as described above.

Write-Ups

The write-up of your work should be in a style suitable for submission as a paper to a conference or a journal on information processing. This should be about $5 \sim 8$ pages in one and half-spaced lines including references, tables, diagrams and figures.

Marking and Contribution of Efforts

The same project mark will be given to all members of the same group. Please ensure that each group member contributes an almost equal effort towards the project.

Mark Distribution

Component	**Issues Addressed**	**Points**
Programs [60]	Input Data Generation	15
	Algorithm-1: MPI Native Program	15
	Algorithm -2 PGAS alternative or One-Sided Comm	30
Report [40]	Abstract + Introduction	5
	Problem Description	5
	High-Level MPI- Algorithm-1.	5
	High-Level Other- PGAS Algorithm-2.	5
	Descript. Exp. Environment	5
	Discussion of Results	10
	Conclusion + References	5