University of the Witwatersrand
School of Electrical & Information Engineering

ELEN4020A
Data Intensive Computing for Data Science
Course Project - Matrix Transposition of Big-Data

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

Due Date: 17 May 2019

***Abstract*—This report discusses project work done on the transposition of 2-dimensional square matrices using straight MPI with derived data-types. The algorithm was done on array sizes of 2^n, where n = {3, 4, 5, 6, 7} and made use of four processors. The inputs were randomly generated between [0, 500]. The algorithm description and its results are present in the paper, further recommendations and a critical analysis is made for the algorithm developed.**

I. Introduction

Matrix transposition is a relatively simple process that can be achieved using basic transposition algorithms. However, these algorithms are not efficient in transposing large matrices. Message Passing Interface is a portable message passing standard which can be used to aid in matrix transposition. This report discusses the development of 2D matrix transposition algorithm which makes use of straight MPI and derived data-types. Included in the document are the background, which includes the project requirements and success criteria, the report also contains an explanation and pseudo code for the algorithm and derived data-types. These can be found in sections II, III and IV respectively. Running times for the algorithm and the analysis of the algorithm can be seen in Section V. Further critical analysis of the algorithm and recommendations for the algorithm and the project can be seen in Section VI and VII respectively.

II. Background

Message Passing Interface (MPI) is a portable message passing standard, it is not a library but rather specifications for what a library should be [1]. MPI is aimed at addressing the message-passing parallel programming model, the interface attempts to be practical, portable, efficient and flexible [1]. Hence, the project will make use of the MPI standardization in order to improve functionality and optimize performance.

A. MPI Pros and Cons

In MPIs attempts to be practical, portable, efficient and flexible, MPI has many pros when it comes to parallel programming, however MPI is limited [2]. Below are the pros and cons for using MPI:

- Pros:
  - Can run on distributed or shard memory architectures
  - Each process has its own local variables
  - MPI can be used for more problems than OpenMP
  - Distributed memory computers are cheaper than shared memory computers
- Cons:
  - Harder to debug
  - MPI performance is limited by the network between nodes

B. Project Requirements

The project has the following requirements; it should be conducted using the Linux programming environment and with the use of one or more of the specified editors. All project documentation is to be done using Latex. The problem addressed is the development of a 2D matrix transposition algorithm using straight MPI with derived data types. The input matrices are to be run for array sizes 2^n, where n = {3, 4, 5, 6, 7}. These will be tested using 4 processors.

To meet the requirements, all the coding is done using Ubuntu 16.04 and with the gedit editor. The use of the cluster, Hornet01, which uses gcc-7.4 and is able to run UPC-2.28.0. The project documentation is done using Overleaf.

C. Success Criteria

The success criteria for the project is:

- Successfully implement straight MPI and transpose a 2D matrix
- The matrix size should be sufficiently large, such as 128×128
- The transposition should work for derived data-types

D. Literature Review

Many algorithms and papers have been done on performing matrix transposition on large data sets with the use of parallel programming architectures. The below articles discuss existing solutions for matrix transposition of large data sets with the use parallel programs:

- The article in [3] discusses parallel matrix transposition algorithms on distributed memory

concurrent processors. The algorithms made use of non-blocking communication between processors to avoid unnecessary synchronization. The processors transpose greatest common divisor wrapped diagonal blocks simultaneously so that the matrix can be transposed with basic lowest common multiple steps, where the lowest common multiple is the lowest common multiple of the block scatter processor template.

- [4] discuses remapping multi-dimensional arrays using a cluster under MPI, OpenMP and hybrid paradigms. It was found that the developed vacancy tracking algorithm outperforms the 2-array method. The paper proved that across the entire cluster of SMP nodes, the hybrid MPI/OpenMP hybrid implementation outperformed the standard MPI by a factor of 4.44.
- An inversion parallel algorithm for adjacent matrices using MPI was developed in [5]. The algorithm was successful in finding inverses for full pentadiagonal matrices. This algorithm with slight adjustments could be used for transposing as inverting matrices also requires remapping.
- The article in [6] analysis MPI derived data-types for numerical libraries. It discussed the derived data-type mechanism of MPI in three examples, one of these examples included matrix transposition. This paper indicates that derived data-type implementation has performance advantages over straight forward hand-written implementations.

III. Explanation of The MPI Transposition Algorithm

The algorithm runs for array sizes of $N = 2^n$ where n = 3, 4, 5, 6, 7. However, for simplicity the algorithm will be explained using a 128 $\times$ 128 matrix. The MPI transposition algorithm makes use of four processes, each process creates a 128 $\times$ 32 sub-matrix and randomly allocates it with elements. This sub-matrix is then split into 1024 sub-sub-matrices with 4 elements each and these elements are then transposed using a function that makes use of the basic transposition algorithm. After this first layer of transposition, the 128 $\times$ 32 sub-matrix is also then transposed. Three out of the four processes then send their respective transposed sub-matrix, using the MPI_Send() function, to the other process which receives the sub-matrices using the MPI_Send(). This process then arranges the four sub-matrices into one 128 $\times$ 128. This text file is then printed into a text file.

IV. Explanation of Derived Data-Types Used

The derived type used is of type MPI_Type_vector(). The datatype has the following properties: it has a count of 128, each block has a block length of one and only stores one element. The *coltype* datatype as shown in Fig. 1, is derived from the primitive datatype MPI_INT.

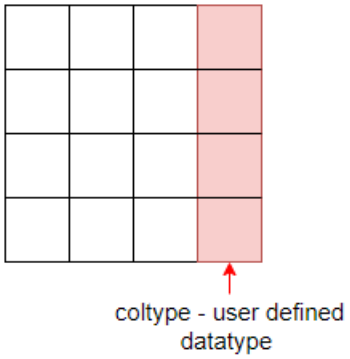

Fig. 1. The communication between the file transfer client and server when a user logs in or registers

The 4 $\times$ 4 matrix in Fig. 1 is only used to illustrate the derived datatype used, the actual datatype used has dimensions of 128 $\times$ 1.

V. Results

Table 1 shows the results for running times of different matrix sizes.

TABLE I
Run Time for Matrices of $N_0 \times N_0$

N_0	Run time (s)
8	1.203
32	2.315
64	4.387
128	9.479

VI. Critical Analysis of the Algorithm

A. Successes

The algorithm successfully transposes matrices of different sizes. It uses four processes to transpose the code.

B. Flaws

One of the flaws of the algorithm is that it only runs and transposes matrices successfully if four threads are used. This is because the transposition algorithm used strongly dependents on the four processes and would not work if less than four or more than four processes are used. Another flaw of the algorithm is that, instead of outputting the entire matrix into a single file, it output the matrix into four different files, one created by each thread. The algorithm also does not take in matrix elements from a text file, rather each thread randomly generates elements for each sub matrix.

The efficiency of the algorithms is determined by analyzing the running time. The running times are shown in table 1.

VII. Recommendations

As one sided communication was not used, in future it is recommended that one sided communication is used for the algorithm. This is due to the advantages of one sided communication, one sided communication can reduce synchronization and improve performance. One sided communication is also able to reduce data movement, two sided communication sometimes adds extra intermediate buffering. One sided communication also can simplify programming and is easier to debug as opposed to two sided communication. Another feature to implement in the future,is running the code on multiple nodes. Running with multiple nodes can avoid data replication and has faster communication. It is also recommend that for working with files in MPI to use the functions MPI_File_read() and MPI_File_write() as it allows for files to be read and wrote with parallel operations.

VIII. Conclusion

Through the use of straight MPI and derived data-types, matrix transposition was successfully implemented for multiple matrix sizes. This was only able to be done using four processors. Based on the algorithm and the results, recommendations were made for future use and implementation. A critical analysis of the algorithm was also made.

References

[1] B. Barney, L. Livemore,"Message Passing Interface (MPI)", Jan. 2019, https://computing.llnl.gov/tutorials/mpi/, date accessed 10/5/2019

[2] Anon, "Pros and Cons of MPI/OpenMP", Dartmouth College, Feb. 2011, https://www.dartmouth.edu/classes/intrompi/parallelprog, date accessed 10/5/2019

[3] Choi, Jaeyoung Dongarra, Jack Walker, David, "Parallel matrix transpose algorithms on distributed memory concurrent computers". 245 - 252. 10.1109/SPLC.1993.365559.

[4] Yun He and H. Q. Ding, "MPI and OpenMP Paradigms on Cluster of SMP Architectures: The Vacancy Tracking Algorithm for Multi-Dimensional Array Transposition," SC '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, Baltimore, MD, USA, 2002, pp. 6-6.

[5] M. E. Kanal, "Parallel algorithm on inversion for adjacent pentadiagonal matrices with MPI," Springer Science, October 2010.

[6] Bajrović E., Träff J.L, "Using MPI Derived Datatypes in Numerical Libraries," Recent Advances in the Message Passing Interface. EuroMPI 2011. Lecture Notes in Computer Science, vol 6960. Springer, Berlin, Heidelberg