ELEN4020A: Out-of-Core Matrix Transposition of Big Data using MPI-IO

Kopantsho Mathafa (849038) Chizeba Maulu (900968) James Phillips (1036603)
May 17, 2019

I. Introduction

Computational operations on data-sets are generally performed using core memory. Very large datasets which exceed core memory pose both a memory capacity and computation challenge. Out of core data, or that stored on a hard disk, solves the memory capacity problem [1]. This solution entails saving the data on a local or remote file which is accessed by the program which contains the algorithm(s) to be applied to the data. However, accessing this file, provided it is large, is computationally expensive if performed sequentially. Collective IO presents an elegant solution to this problem by accessing the data using multiple processes which divide the data among themselves [2]. Once accessed, this data need be operated upon by these processes. This entails transfer of data between these processes. One- or two-sided communication can be used depending on whether both processes are required to participate in the communication or need be synchronized.

This paper presents the transposition of an out-of-core matrix using one-sided communication functionality and dervided datatypes defined in Message Passing Interface (MPI). MPI is a standard interface used to write distributed memory programs [3].

The rest of the paper is organized as follows, Section III presents the problem under study. A brief literature review is performed in Section II. The implemented MPI program as well as the experimental environment in which it is run used is described in Section IV. Section V critically evaluates the results of the algorithm.

II. Literature Review

Fu et al. propose a framework based on MPI one-sided communication for graph processing [1]. Given the size and rapid growth of the graphs, they are difficult to store on a single machine. The framework is formulated accross the global address space. It uses MPI `Put` to transfer data to enhance scalability and efficiency. The computation processes is divided into portions that can be overlapped. The result is the framework has 35% faster job completion time than frameworks that employ two-sided communication to the same problem. He et al. apply one-sided communication to the computationally-intensive Computed Tomography (CT) image reconstruction process [4]. The process is used to deduce a patient's internal structure using a CT scans projections. The underlying algorithm used in the process is the Katsevich algorithm.

Zhou et al. employ the one-sided communication functionality defined in MPI-2 with an asynchronous engine for non-blocking communication operations [5]. A scheme is developed that uses an arbitrary amount of processes with to drive asynchronous progression as opposed to an additional thread within the application. The scheme intends to improve the communication cost between processes and ensure efficient communication is overlapped efficiently.

Similarly, Dinan et al. develop an MPI-based runtime for programs with Partitioned Global Address Space (PGAS) models [6]. They assert that the one-sided communication defined in MPI-2 does not sufficiently support higher-level global array parallel programming models. Their design enhances interoperability and portability of MPI and GA applications.

III. Problem description

A group of processes is required to generate a large, two-dimensional square matrix. The elements of this matrix are randomly generated integers with range 0-99. This matrix must be written to a file which will serve at the input file to the transpose algorithm. The format of this file is required to have the rank of the matrix as the first value; the rest of the values are the elements of the matrix in row-major order starting from $A[0][0]$, where A is the matrix. This matrix is the global input matrix to the program. Reading the data requires dividing the matrix between several processes. These processes will then each have a sub-matrix of the size defined by the programmer. Each process will perform the transposition. Once complete, the transposed sub-matrices are assigned to different processes. The

transpose must be written to an output file in row-major order. The program must be tested with matrix sizes of 2^n where n = 3,4,5,6,7. The number of processes P should have values P = 16,32,64.

IV. Algorithm

The program is run on an 8-core machine with 16 GB of memory. The CPU has a clock rate of 3.4GHz.

V. Evaluation of results

VI. Conclusion

VII. Pseudocode

A. Word count algorithm

Data: Any data type object and a container
Result: An intermediate key/value pair
Convert all characters in objects text data to uppercase;
while *While not at the end of the text data* **do**
  while *While not reading an uppercase char alphabet* **do**
    Increment a counter;
    If the counter has increased in value Store the word;
    if *If it the word is a stop word* **then**
      Do nothing;
    else
      Send out intermediate key/value pair for further processing;
    end
  end
end

Algorithm 1: Phoenix++ map function implementation

B. Inverted Index algorithm

Data: K value
Result: List of top K occuring words and the lines they appear in
for *From 0 to K* **do**
  for *From 0 until the end of the text data* **do**
    for *Each non-stop word found in the text* **do**
      Store the line which the word is found on;
    end
  end
end
Print the top most appearing words and the lines they are found on;

Algorithm 2: Inverted index function implementation

VIII. Conclusion

The MapReduce solution of the word count, top-K query and inverted index problems is presented. MapReduce decomposes large datasets and divides their reading and processing between processing units. The MapReduce functionality of the Phoenix framework is utilised in the solution. The word count algorithm of the large dataset takes longer than that of the small dataset. The top 10 and 20 queries are determined in relatively similar times for both datasets. Although a serial inverted index algorithm is implemented, the MapReduce algorithm was unsuccessful. No running time tests are performed due to this.

References

[1] H. Fu, M. Gorentla Venkata, S. Salman, N. Imam, and W. Yu, "Shmemgraph: Efficient and balanced graph processing using one-sided communication," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2018, pp. 513–522.

[2] S. Seo, R. Latham, J. Zhang, and P. Balaji, "Implementation and evaluation of mpi nonblocking collective i/o," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2015, pp. 1084–1091.

[3] Byna, Gropp, Xian-He Sun, and Thakur, "Improving the performance of mpi derived datatypes by optimizing memory-access cost," in *2003 Proceedings IEEE International Conference on Cluster Computing*, Dec 2003, pp. 412–419.

[4] T. He, J. Ni, J. Deng, H. Yu, and G. Wang, "Deployment of one-sided communication technique for parallel computing in katsevich ct image reconstruction," in *First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06)*, vol. 1, June 2006, pp. 416–423.

[5] H. Zhou and J. Gracia, "Asynchronous progress design for a mpi-based pgas one-sided communication system," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2016, pp. 999–1006.

[6] J. Dinan, P. Balaji, J. R. Hammond, S. Krishnamoorthy, and V. Tipparaju, "Supporting the global arrays pgas model using mpi one-sided communication," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, May 2012, pp. 739–750.