Some title

ELEN4020

1st Darren Blanckensee
School of Electrical and Information Engineering
University of the Witwatersrand
Johannesburg, South Africa
1147279@students.wits.ac.za

2nd Uyanda Mphunga
School of Electrical and Information Engineering
University of the Witwatersrand
Johannesburg, South Africa
1168101@students.wits.ac.za

3nd Ashraf Omar
School of Electrical and Information Engineering
University of the Witwatersrand
Johannesburg, South Africa
710435@students.wits.ac.za

4ndAmprayil Joel Oommen
School of Electrical and Information Engineering
University of the Witwatersrand
Johannesburg, South Africa
843463@students.wits.ac.za

***Abstract*—.**

I. Introduction

TWO parallel join algorithms were implemented and tested on two different parallel frameworks. The implemented algorithms are the join and hash joins. The hash join algorithm was implemented on MPI and the merge join algorithm was implemented on openMP. The purpose of the project are twofold, the primary task is to use parallelism to implement two different join algorithms and the secondary is to compare the algorithms' efficiency in joining. This project discusses existing solutions of other authors and the context in which this project was implemented. This report also contains sections on the Design and Implementation of the codes, Testing and Results, a SWAT analysis of the algorithms and possible Future Improvements in implementing the algorithms.

II. Background

The join algorithms are designed to read data from file of a specific format and to perform the required operations as per specific algorithm. The algorithms read data of the same file in order to be able to compare their performance. The algorithms are both implemented in C++. This allows for accuate comparisons of the algorithm performances and consequently prevents the possible time lags that may arise from the use of different programming languages. C++ was used instead of C due to the authors finding easier to use C++ under the given time constraints. Another reason that why C++ was found more favourable is that is offers more functionality than C and due to its Object Orientated Design (OOD) capabilities and the also due to the fact that the authors are familiar with C++. These algorithms were implemented using parallelism.

A. Existing Solutions

Based on the literature, there are multiple join algorithms. The algorithms chosen by the authors are the hash algorithm and the merge algorithm, which form part of the equi-join family of join algorithms and are widely used [1].

B. Literature Review

This section views some of the literature on the has and join algorithms.

1) Merge Join Review: References [1], [2] take into account possible errors that may occur due to data skew for the merge join algorithm. According to [1], data skew can result in some of the processors being over-utilised and others under-utilised. Reference [1] proposes the use of a divide-and-conquer technique as a solution to handle data skew. Reference [2] approaches the issue of data skew in the merge algorithm by adding an extra scheduling phase. The implemented merge join does not take into account the possibility of data skew and instead relies on openMP's computational ability to process certain chunks of data and ability to make numerous threads [3]. This technique was chosen due

to the resources available, namely a cluster, and time constraints.

2) Hash Join Review: The reviewed literature for the hash join is also concerned about data skew. Some of the said literature include references [4], [5] and other documents. Reference [4] addresses the issue of data skew by implementing multiple hash phases along with the use of a heuristic optimisation algorithm in order to isolate skew elements [4].

C. Merge Join

Merge join, as stated above, is also known as sort-merge join [6]. The basic idea of this algorithm is to sort data (in parallel in this context) according to the attributes that are going to be used in the joining process [7]. The implemented algorithm assumes that the sorting of the data has already taken place and proceeds to only perform the join algorithm under that assumption.

D. Hash Join

A hash join can be generalised into two stages [5], the split phase and the join phase [8].

E. Assumptions

It is assumed that the data read has no errors and no error checks are provided.

F. Constraints

The experienced constraints included the allocated time to complete the project and the delays that were experienced due to technical issues, such as the malfunctioning of the cluster.

G. Success Criteria

Successful implementation of parallelism for the join algorithms.

H. Time Management and Work Division

——————someone put a table here——————-

III. Design and Implementation

A. Merge Join

In the implementation of the merge join, classes were used in order to attempt to follow an object-orientated paradigm. The code is designed to function in three steeps. The first step is that it reads the first table's input data. In the second step it reads from the second file. In the third step the code then implements the join algorithm. The code works by comparing the key values of the input data and loads them into containers if the key values are the same. The code is designed to also match the data values even if the key values of the input values are duplicated, provided that the repeating keys are grouped together.

B. Hash Join

IV. Testing and Results

The merge join algorithm works for input consisting of values around 25 lines or less. In the instance that one has input lines that go above a certain threshold errors occur. Due to this, it is rendered impossible to run the code for the analysis of very large file.

V. SWAT Analysis

A. Strengths

B. Weaknesses

The merge join code crashes after a certain number of input lines of code is reached. The code also violates the DRY (Don't Repeat Yourself) principle by having multiple classes which essentially perform the same tasks. Due to time constraints ad delays that the code for merge join is flawed and requires time to debug.

C. Advantages

D. Threats

VI. Future Improvements

The merge join code should be re-made with the use of minimal amount of classes and vectors.

VII. Conclusion

References

[1] J. L. Wolf, D. M. Dias, and S. Philip S. Yu, "A parallel sort merge joinalgorithm for managing data skew," IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, Tech. Rep., 1993.

[2] J. L. Wolf, D. M. Dias, and P. S. Yu, "An effective algorithm for parallelizing sort merge joins in the presence of data skew," IBM Research Division. T. J. Watson Research Center, Tech. Rep., 1990.

[3] B. Barney. Openmp. Lawrence Livermore National Laboratory. [Online]. Available: https://computing.llnl.gov/tutorials/openMP/

[4] J. L. Wolf, D. M. Dias, P. S. Yu, and J. Turek, "An effective algorithm for parallelizing hash joins in the presence of data skew," IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, Tech. Rep., 1991.

[5] A. M. Keller and S. Roy, "Adaptive parallel hash join in main-memory databases," Stanford University and Advanced Decision Systems, Computer Science Department, Shell Development Company, Tech. Rep., 1991.

[6] G. Graefe, "Sort-merge-join: An idea whose time has(h) passed?" Portland State University, Tech. Rep., 1994.

[7] ——, "Heap-filter merge join: A new algorithmfor joining medium-size inputs," IEEE TRANSACTIONS ON SOFIWARE ENGINEERING, Tech. Rep., Sep. 1991.

[8] M. KITSUREGAWA, S. ichiro TSUDAKA, and M. NAKANO, "Parallel grace hash join on shared-everything multiprocessor :implementation and performance evaluation on symmetry s81," Institute of Industrial Science, University of Tokyo7-22-1 Roppongi, Minato-ku, Tokyo, 106 Japan, Tech. Rep., 1992.