# **COMP 6521 LAB ASSIGNMENT 1**

Sahaja Gottipati 40093560

Margaux 40151012

Suseela Reshma Avireddy 40105836

# **INTRODUCTION**

This document gives a brief on the Two-Phase Multi Way Merge Sort (TPMMS) method and the implementation methodology. TPMMS is the process of sorting a input file of tuples. This involves the task of merging multiple file streams into a single sorted file. This document also includes a theoretical explanation of the method with detailed analysis of the test cases/results and the evaluation of the performance.

# **TPMMS ALGORITHM:**

To be able to merge and sort the 2 list we used the **Two-phase Multiway Merge-sort**. This method implies 2 phases:

1. Sort main memory-sized partitions to produce sorted sublists (=runs)
2. Merge the sorted sublists into single sorted list

Detailed descriptions of the 2 phases is as follows:

The available main memory must be filled with blocks to be sorted. We can sort them in the main memory using techniques such as quicksort. Subsequentially the sorted records must be written from the main memory to “new” blocks of disk. This will result in sorted sublists [1]. To summarize 3 steps will be conducted:

1. Divide the input file into chunks of M blocks each
2. Sort each chuck individually using the M buffers
3. Write the sorted chunks to disk [2]

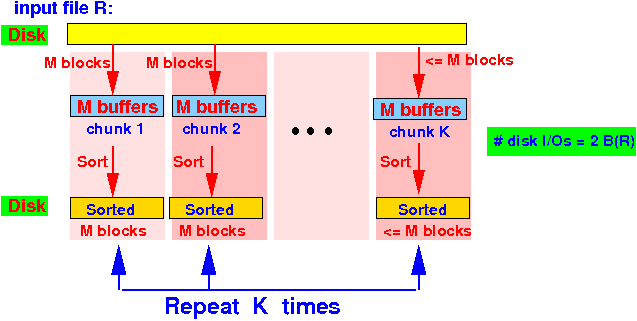


Figure 1: Graphical representation phase 1

In the second phase of the process the sorted shortlists must be merged into a single sorted list. The amount of passes in this phase depend on the size relation (R), size of the tuple (N), the block size(B) and the main memory input buffers (M). Since we need one output buffer we can use (M/B)-1 buffers for the process with M/B the amount of main memory buffers.

The runs can be calculated by r = NR/M and we can merge M/B-1 runs. In other words after one pass we have r/((M/B)-1) runs and after k runs r((M/B)-1)k. The phase is conducted until there is only one single run left. This implies the following: r/((M/B)-1)k = 1. [1]

To make it more explicit a graphical representation will be given where the following steps are executed:

1. Divide the M buffers into: M-1 input buffers and 1 output buffer
2. Use the M − 1 input buffers to read the K sorted chunks (1 block at a time)
3. Merge the K sorted chunks together into a sorted file using 1 output buffer as follows:
   * Find the record with the smallest sort key among the K buffers
   * Move records with smallest key to the output buffer
   * When output buffer is full, write output buffer to disk
   * When some input buffer is empty, read another block from sorted chunk if there is more data [2]

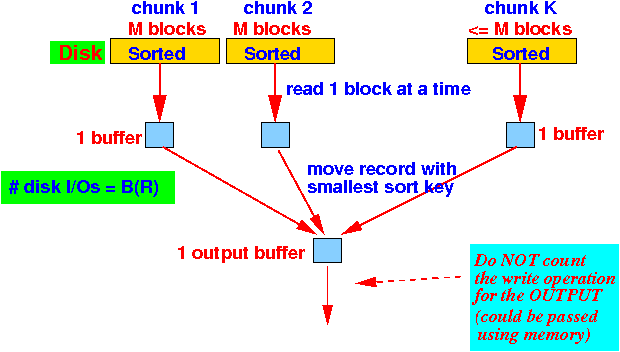


Figure 2: Grapical representation of phase 2

# **IMPLEMENTATION OF THE ALGORITHM:**

In phase one the input files T1 /T2 are read sequentially. The complete data will be read based on the block size (based on main memory). For example, when you have 10 tuples and the main memory only can read 5 tuples, we will need 2 run to complete the reading. After sorting is implemented for the respective chunk of data, it is written to n batch text files.

In phase two all the temporary batch files are merged and the sorted according to the employee ID and each tuple is checked for duplicate values. In presence of duplicates the tuples are checked against each other for the latest updated data and updated the output file accordingly. For instance, if there are two tuples containing the same employee ID then the date column is checked and the latest date tuple is taken into consideration.

More concretely the following steps will be carried out for T1 and T2:

1. Read data in blocks
2. Write individual tuples into files without duplicates for employee ID’s and sorted on the employee ID’s (take updated date)
3. Merge two files and check for duplicate employee ID’s . Take updated date for employee ID and write it to a file
4. Repeat step 3 until a single output is obtained

Finally, the obtained T1 and T2 single output file is merged with the duplicates values removed and updated according to the latest date.

A visual representation is given below:

As example we assume for T1 10,000 tuples, T2 20,000 and a block size of 5,000. This will lead to the following process flow:

Comparative

Figure 3: Graphical example implementation algorithm

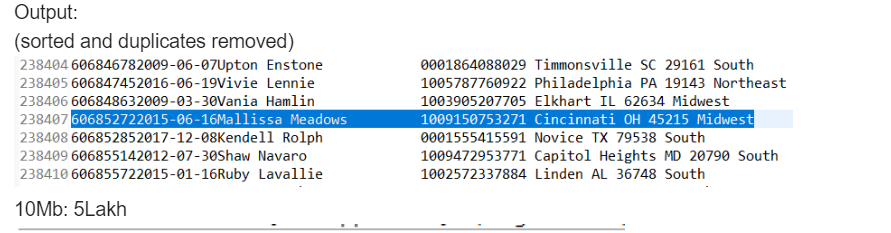
In the implementation of the algorithm following definitions are used:

* One memory fill (block size) = main memory/4096
* Numbers of tuples in one memory fill = (block size \* 4096)/100
* Number of blocks = (no of tuples of data(size of data)\*100)/4096
* Number of rounds to process whole data is = ceil (number of blocks/ block size)

# **TEST RESULTS/CASES:**

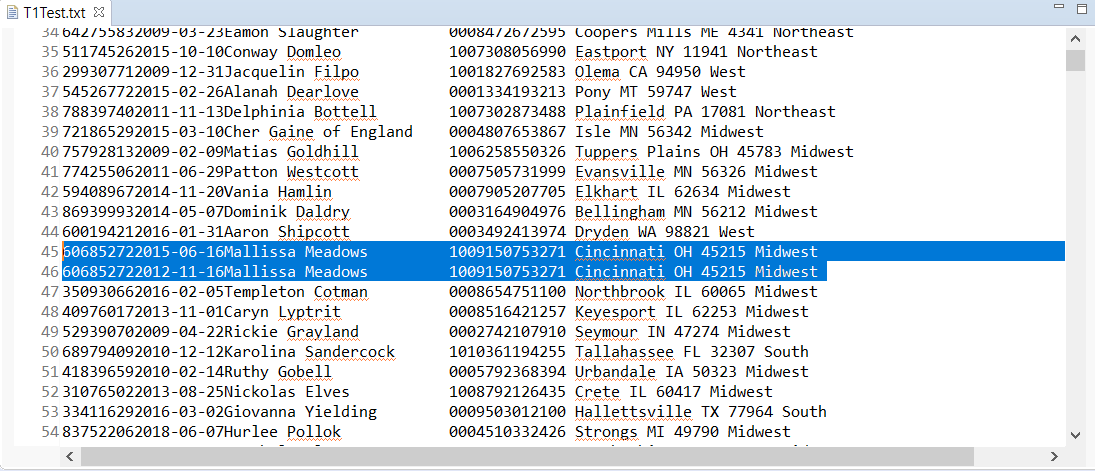
The following test cases were evaluated:

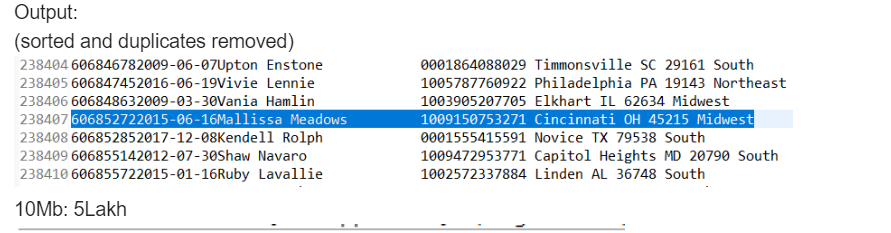
1. The files are sorted according to the employee ID:



*Fig 4: Sorted employee ID*

1. Data is checked for duplicates, removed and updated according to the date.



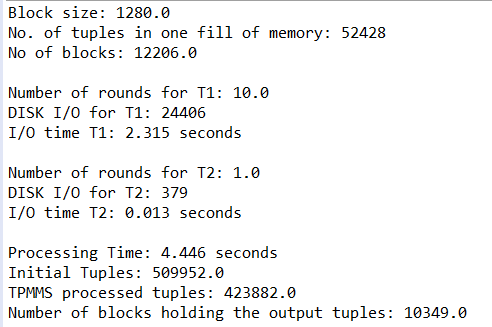


*Fig 5: Data duplicates checked and removed*

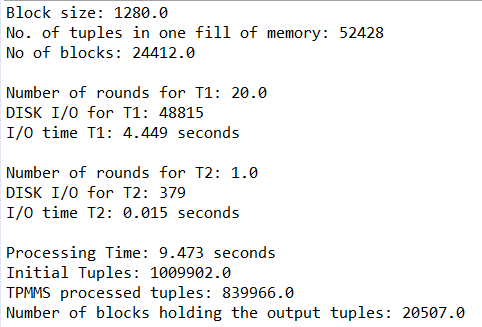
# **EVALUATION AND PERFORMANCE:**

Evaluation and performance are based on memory constraints:

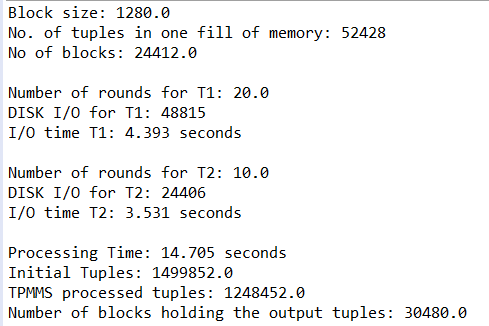
1. 10MB:



*Fig 6:10MB evaluation with 1 round 5 lakh data*

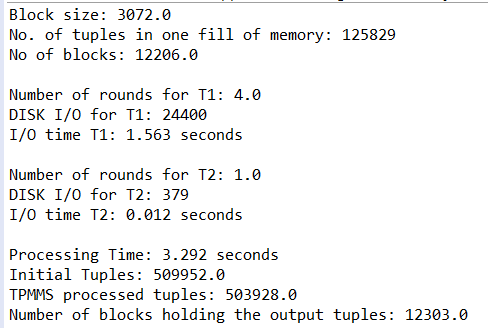


*Fig 7:10MB evaluation with 10 rounds 10 lakh data*

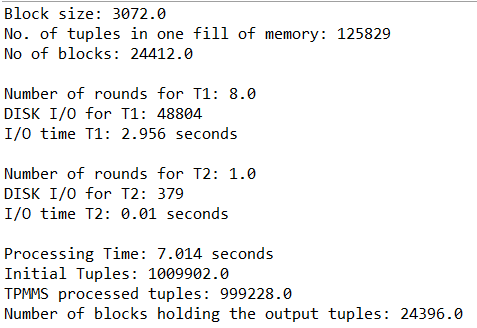


*Fig 8:10MB evaluation with 1 round 15 lakh data*

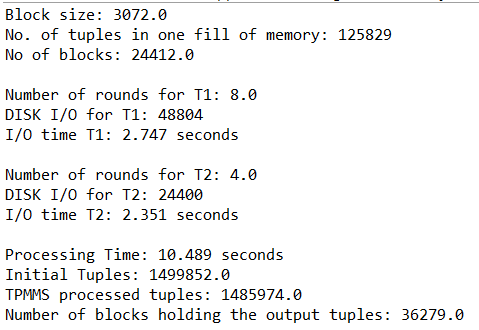
1. 20MB:



*Fig 9: 20MB evaluation with 5 lakh data*



*Fig 10: 20 MB evaluation with 10 lakh data*



*Fig 11:20 MB evaluation with 15 lakh*

**ANALYSIS OF NUMBER OF BLOCKS :**

|  |  |
| --- | --- |
| **Memory** | **No of Blocks Holding O/P tuples** |
| 10 MB – 5 lakh data | 10349 |
| 10 lakh data | 20507 |
| 15 lakh data | 30480 |
| 20 MB – 5 lakh data | 12303 |
| 10 lakh data | 24396 |
| 15 lakh data | 36279 |

**TEAM CONTRIBUTION:**

* Sahaja Gottipati And Margaux: Implemented the read and write phase with block division, memory calculation and implemented the sort and removal of duplicates of one chunk of data. Been part of Documentation of the report.
* Suseela Reshma Avireddy and Margaux: Implemented the merge phase with removal of duplicates. Also been part of implementation of the report.

# **BIBLIOGRAPHY**

|  |  |
| --- | --- |
| [1] | H. Garcia-Molina, J. D. Ullman and J. Widom, Database Systems: The Complete Book, New Jersey: Pearson Education, 2009. |
| [2] | "The multi-pass multiway merge sort algorithm," [Online]. Available: http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/4-query-exec/multi-pass=sort.html. |