Report

for

Programming Assignment 2a

Manthan Patel
(A-20413535)
mpatel120@hawk.iit.edu

Index

1. Introduction

Sorting is the mechanism by which we can arrange the data in ascending or descending order of the given values (if values are non-numeric then the sort will be based on the ASCII value of the characters). It is easy to sort the data if the data size is less than the in-memory(RAM) size as we can sort the data within the memory using the algorithm which is best suitable in terms of time and memory (like MergeSort, QuickSort, HeapSort). There might be possible that the given unsorted data which is higher than the memory size. At that time External sort method can be used to sort the whole data.

External sorting is the method in which the input data gets divided into the files which has size nearly equal to the memory size (for example 80% of memory size) so that we can maximize the use of memory while sorting the data and create less number of files. After sorting the data into the files, we need to use k-way merging to sort the data taking from the sorted files into small chunks (but big enough so that total of all chunk sizes is 70% - 80% of the memory size) and then sort/merge into an output file.

I have used the QuickSort algorithm (pivot at last point) for sorting the data using multiple threads (1, 2, and 4) for both the files 2GB and 20GB. For 2GB file, the sorting will be done within the memory as the total memory size is more than 2GB. However, for 20 GB data I have created 10 files, each having 2GB of sorted data, which will be used for final merge of 20GB file. I have taken 500MB of data from each sorted file and merged them into 1GB of an output buffer and finally saved it in output file.

In 2GB of input file, I have created 4 threads, each having 500MB DATA, and then after completion of sorting I merged all the data into single buffer (in main method) and copied it in output file.

In 20GB input data, I have used 3 threads, each having 2GB data to sort and save in file, and one shared variable which will have track of number of file generated. Every time thread will check this file counter and if it is not at maximum (for 20GB max 10 files), then thread will do sorting again with the new 2GB of data and will save it in new file. Once the files will be created, the threads will be destroyed and merging (using k-way merge) of data will start in sequential way. The buffer size for each file 500MB of data from each file and 1GB of output buffer.

I have managed all the variables' value assignment based on 3 main variables (which I have defined in the beginning of the program). They are **Total number of files** that we want to create for external sort, **Input file size** and **Maximum buffer size** for k-way merge. I have also defined the Thread count at the beginning.

2. Steps to Run the Project

The external sort or Terasort has been written in C language and it has used POSIX thread library for multi-threading.

Commands for compilation and running project

To compile the .c file on terminal the command is -
For compilation of the file for 2GB data sort-

```
gcc -Wall mySortBenchmark_2GB.c -o mySortBenchmark2 -lpthread
```

and for compilation of the file for 20GB data sort-

```
gcc -Wall mySortBenchmark_20GB.c -o mySortBenchmark20 -lpthread
```

Finally, to run the code after successful compilation, the command is -
for 2GB data sorting method-

```
./ mySortBenchmark2
```

for 20GB data sorting method-

```
./ mySortBenchmark20
```

Using Makefile

I wrote a Makefile which includes the command for remove object and executable files and then compile both the .c files. The command to run the makefile is-

```
make all
```

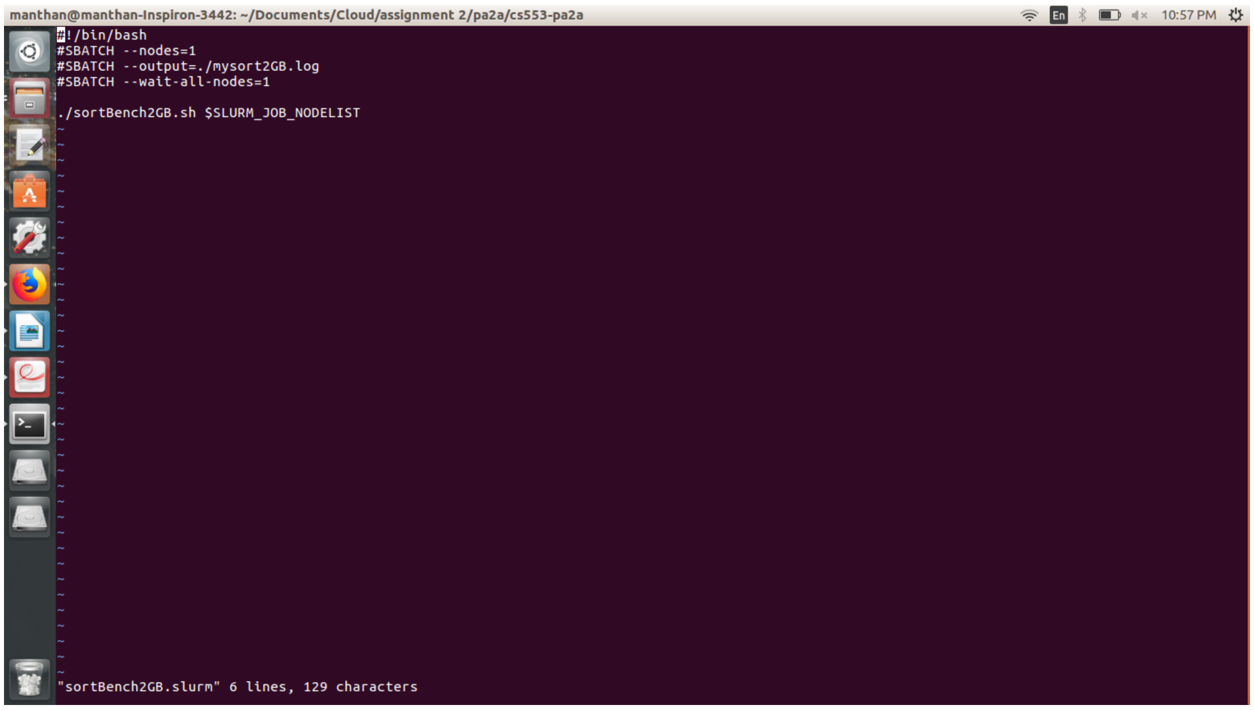
SLURM Job command

For testing my program in **neutron** cluster, I have written 4 SLURM scripts and 4 shell scripts which will do the running and then testing (by valsort) the output file tasks.

To run the slurm script the command is-

```
sbatch linSort2GB.slurm
sbatch linSort20GB.slurm
sbatch sortBench2GB.slurm
sbatch sortBench20GB.slurm
```

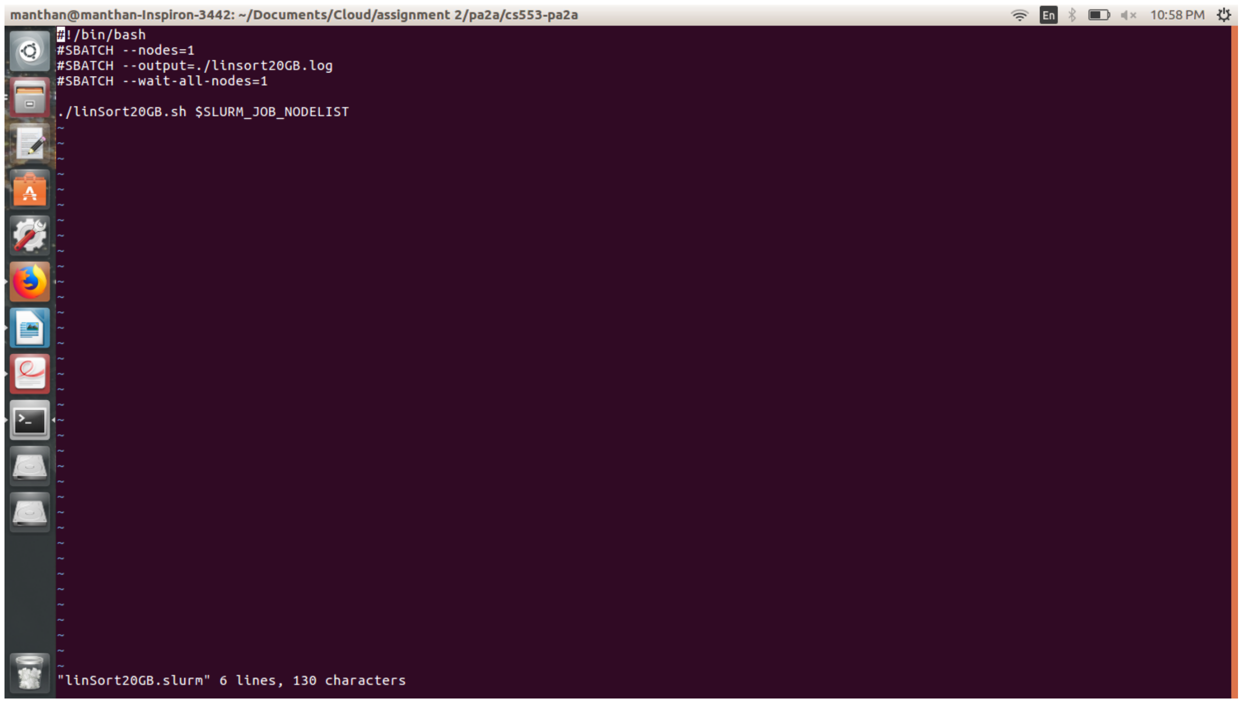
one of the slurm script is given in the bellow image-


manthan@manthan-Inspiron-3442: ~/Documents/Cloud/assignment 2/pa2a/cs553-pa2a

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --output=./mysort2GB.log
#SBATCH --wait-all-nodes=1

./sortBench2GB.sh $SLURM_JOB_NODELIST
"sortBench2GB.slurm" 6 lines, 129 characters
```

Fig.1 SLURM file for **“sortBench2GB.slurm”**


manthan@manthan-Inspiron-3442: ~/Documents/Cloud/assignment 2/pa2a/cs553-pa2a

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --output=./linsort20GB.log
#SBATCH --wait-all-nodes=1

./linSort20GB.sh $SLURM_JOB_NODELIST
"linSort20GB.slurm" 6 lines, 130 characters
```

Fig. 2 SLURM file for **“linSort20GB.slurm”**

The output will be saved in-

- ***mysort2GB.log***
- ***mysort20GB.log***
- ***linsort2GB.log***
- ***linsort20GB.log***

3. Output and Performance

Output Table

I have taken the output for the Linux sort function as well as output of my sorting function for 2GB file and 20GB file. I have also tested my function for 10GB of data (read from 20GB file) with 4 threads.

For 2GB file, I have tested my function with 1 and 4 threads. The output from that I received from it is as below in table-

Experiment	Shared Memory (1VM 2GB) 4 Threads	Linux Sort (1VM 2GB)	Shared Memory (1VM 20GB)	Linux Sort (1VM 20GB)	Shared Memory (1VM 2GB) 1 Thread
Compute Time(sec)	39.711521	23	797.900024	352	32.174095
Data Read(GB)	2	2	40	40	2
Data Write(GB)	2	2	40	40	2
I/O Throughput(MB/sec)	100.7264365	173.913043	100.2631879	227.272727	124.3236212

Table 1. Performance evaluation for TeraSort

From the Table 1, the performance of the TeraSort function gets decrease as I have increased the threads. That is because even if we sort the data using threads we need to do final merge in sequential way which will take extra time for comparisons. Hence the performance gets degraded. Whereas for 20GB we are already going to do the merge part after sorting data in small files (for example 2GB file) hence the threading will definitely increase the performance with the increase in number of threads.

If we compare the 2GB sorting methods i.e., TeraSort method and Linux sort method, then I can say that I have achieved the compute time efficiency for TeraSort function is nearly equal to 60% using 4 threads and 70% using single thread.

For 20GB sorting methods, the compute time efficiency for TeraSort function is nearly equal to 45%.

NOTE: Data Read and Write operations for 20GB file will be 2 times for each. For the first time I am sorting the data having 2GB size and saving it in a file and total 10 files will be there. Finally, I am taking 500MB data from each file and writing it in final sorted output file. I am assuming that the Linux Sort function is also creating total 20GB small chunk files as to do external sorting. Therefore,

Total read operations = 20GB (main input file) + 20GB (2GB data of 10 sorted files)
Total write operations = 20GB (final output file) + 20GB (2GB data of 10 sorted files)

The formula used for the calculations is-

I/O Throughput (in MB/sec) = (Total data read + Total data write)/Total time taken

Graphical representation

Based on the output that I received for my external sorting benchmark and Linux sorting function, I have made 2 comparisons for 2GB and 20 GB data set files. The bar graphs for these comparisons are given below-

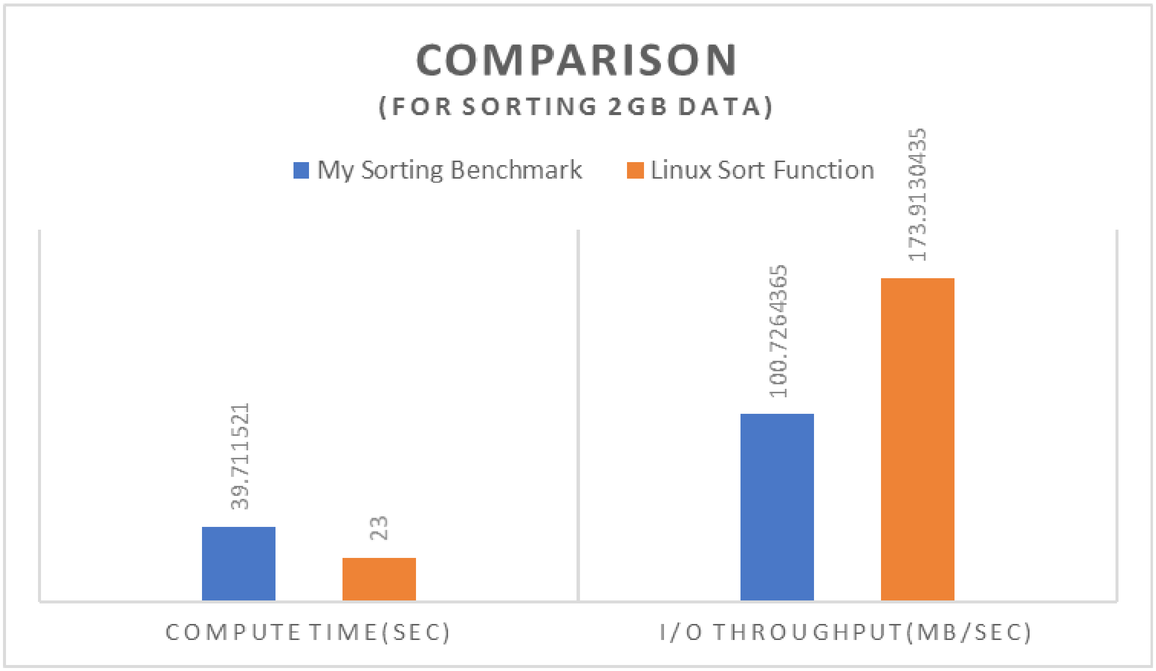


Fig. 3 Comparisons for 2GB data sorting method

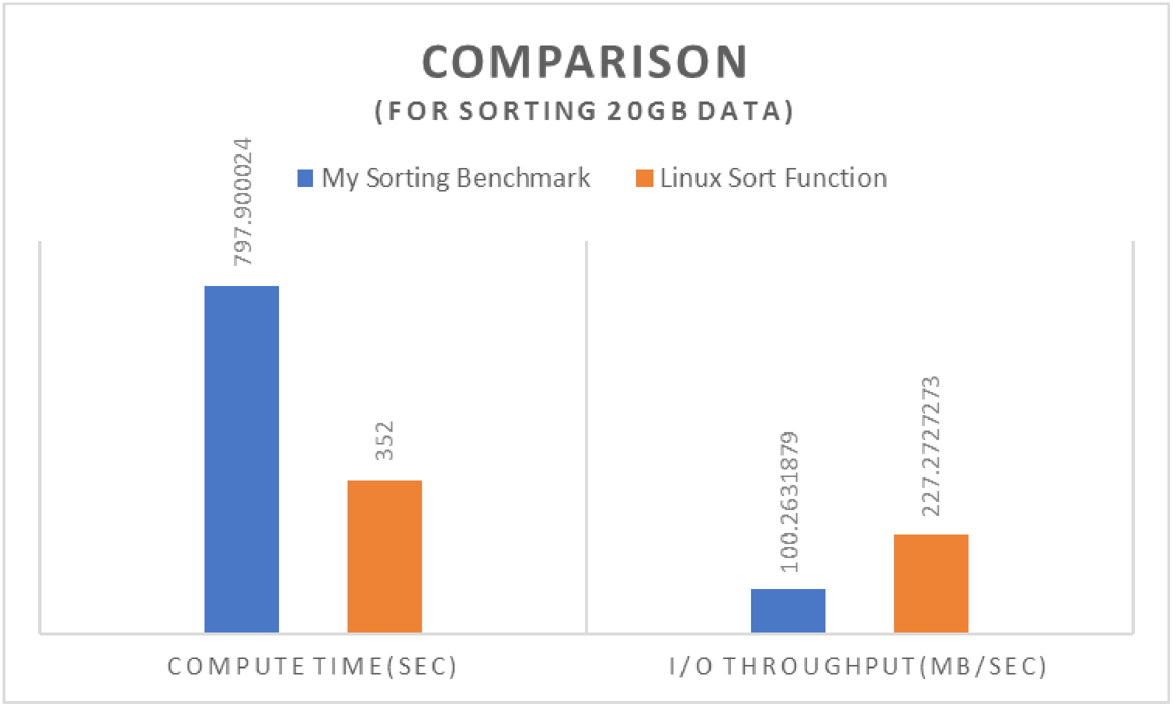


Fig. 4 Comparisons for 20GB data sorting method

Fig. 3 and 4 shows the comparisons between Linux sort method and TeraSort method for compute time and I/O throughput for the input files of size 2GB and 20GB respectively.

4. Area for Improvement

For 2GB data sorting method, we can use macros instead of creating functions for swapping strings. There might a need of better merging method after sorting the chunks of data into threads. Otherwise, we can use the better version of quicksort which takes uses less operations.

For 20GB data sorting method, there is a need for change in better version of QuickSort or other algorithm which is better in terms of least memory usage and better running time for example heap sort (However, it is not a stable one). There is also one more area that is merging of data using k-way merge. In k-way merge, there is a big part which can affect the performance is allocating memory to each chunk as well as output buffer in such way that the I/O operations get reduce and save more data in memory for comparisons.

5. Conclusion

From the Performance evaluation table and graphs, I can say that I have achieved around 60% to 70% efficiency in terms of compute time for 2GB data sorting (using different number of threads) whereas my sorting method is approx. 45% efficient in terms of compute time. Therefore, there is a need for the improvement for k-way merge using parallelization.

6. Reference

Following are the links for the references taken for the program-

1) **Quick sort Algorithm**- *Introduction to Algorithms* by Charles E. Leiserson, Clifford Stein, Ronald Rivest, and Thomas H. Cormen (Page- 171)
2) **K-way merging**- https://en.wikipedia.org/wiki/K-way_merge_algorithm
3) **External sort**- https://en.wikipedia.org/wiki/External_sorting