Practical Assignment 3

Student Name:
Heyt Gala (hgala@hawk.iit.edu)
Vijay Bharvad (vbharvad@hawk.iit.edu)

Problem Statement:
To sort data of different sizes using the built-in Linux sort command and my sort program using p-threads and C programming language.

Environment:
All the below experiments are run on a local UBUNTU system with configuration as 16 GB RAM and 8 cores (16 threads).

Design:
The C program does one of two types of sorting (Internal sorting or External sorting) depending on whether the huge, unsorted data can fit in memory. Internal sorting is utilized if all the data fits in memory; otherwise, an external sorting algorithm called K-Way External Merge is used to sort the data and merge the files. The software creates a sorted file that may be used to validate the validity of the file by passing it to a valsort. The gensort program is used to produce the input data. The code was created utilizing a multithreaded method to increase IO and compute efficiency.

The following assumptions are made:
1. It is believed that there is only 8GB of usable RAM. As a result, we can externally merge sort datasets larger than 8GB.
2. The same number of threads are used for compute/sort operations and IO operations.

Internal Memory Sort:
We are using multithreading to read the input file into memory. The internal file is further divided into smaller blocks and a quick sort algorithm is used on these smaller pieces to sort them parallelly. All blocks are then merged into one sorted output file once completed. The merging process works as follows:

1. Compute the minimum value from each of the chunks.
2. Write the minimum value into a file.
3. Increment buffer size where the minimum record is found.
4. Repeat the above steps until all the records are merged.

External Sort:
The algorithm sorts the data divided in chunks that are present in RAM and merges all the sorted data chunks together to create a new sorted file. The 2-way merge happens in 2 phases.

Phase 1:
Splitting phase

1. Compute the data that can be fit in memory.
2. Read the data into memory using a multithreaded approach.
3. The in-memory data is then divided into smaller chunks for parallelly sorted using quicksort.
4. Each of the sorted chunks is written to a temporary file.
5. Repeat until the entire file is read, sorted, and written to temporary files.

Phase 2:
Merge phase

1. Compute the memory allocation for each chunk.
2. Read the chunk into memory.
3. Find the minimum record from all the chunks by comparing the first record of each chunk.
4. Write the minimum record into the output file.
5. If the chunk is sorted, get the next chunk in memory.
6. Repeat steps 2-5 until all chunks are read and merged.

Given below is the comparison of the performances of shared memory sort and Linux sort using the given parameters:

1. Memory is limited to 8GB.
2. Multithreading is implemented.
3. Sort algorithm and sort approach.

Results:

Experiment	Shared Memory (1GB)	Linux Sort (1GB)	Shared Memory (4GB)	Linux Sort (4GB)	Shared Memory (16GB)	Linux Sort (16GB)	Shared Memory (64GB)	Linux Sort (64GB)
Number of Threads	8	8	8	8	8	8	8	8
Sort Approach (eg. In-memory / external)	In-memory	In-memory	In-memory	In-memory	External	External	External	External
Sort Algorithm (eg. Quicksort /merge sort/etc.)	Merge sort	Timsort is a sorting algorithm derived from merge sort and insertion sort	Merge sort	Timsort is a sorting algorithm derived from merge sort and insertion sort	Quick sort	Timsort is a sorting algorithm derived from merge sort and insertion sort	Quick sort	Timsort is a sorting algorithm derived from merge sort and insertion sort
Data Read (GB)	1	1	4	4	16	16	64	64
Data Write (GB)	1	1	4	4	16	16	64	64
Sort Time (sec)	1.022	0.372	5.43	1.971	1104	1669.825	5438.94	6020.40
Overall I/O Throughput (MB/sec)	298.69	308.58	322.54	300.456	229.454	256.345	189.343	169.35
Overall CPU Utilization (%)	78	76.67	73.576	74.76	80.45	72.78	76.79	81.68
Avg Memory Utilization (GB)	1.165	1.68	4.61	5.27	1.49	7.21	1.32	7.54

Conclusion:

We can observe from the table that for 16GB and 64GB, MySort.cpp outperforms Linuxsort. Linux sort, however, has an advantage for files up to 1 GB and 4 GB in size. The My Sort. CPP's performance can be enhanced, for example, by adopting better memory allocation and increasing overall code effectiveness. Additionally, CPU usage is above as we use numerous threads when using external sort. But the table below shows utilization as a percentage of 100.

Additionally, inside 8 GB, the average memory consumption is relatively low. When compared to how much RAM Linux sort uses, this is quite little. Most of the time, the total I/O throughput is in the range of 250 MBPS. When merging all of the chunk files in phase 2, the quick sort with merge sort method is employed. My sort was tested using various numbers of threads, and 48 threads proved to be the most effective. All of our observations are contained in the test’s directory. Additionally, the 48 threads make sense given that the Skylake system has 48 logical cores, which allows each thread to execute on a single logical core for the highest performance possible.