CS553 PA #3

Sort on Single Shared Memory Node

Instructions:

- Assigned date: Wednesday October 11th, 2023
- Due date: 11:59 PM on Monday October 23rd, 2023
- Maximum Points: 100%
- This homework can be done in teams of up to 2 students
- Please post your questions to BB
- Upload your assignment on the Blackboard with the following name: Section_LastName1_FirstName1-Section_LastName2_FirstName2.HW1.zip
- Late submission will be penalized at 10% per day.
- Usage of online resources is allowed if citation is included. Referring to AI tools are allowed; you are responsible for the correctness of your submitted code.

1. Introduction

The goal of this programming assignment is to enable you to gain experience programming with external data sort and multi-threaded programming.

2. Your Assignment

This programming assignment covers the external sort (see https://en.wikipedia.org/wiki/External_sorting) application implemented in a single node shared memory multi-threaded approach.

You can use any Linux system for your development, but you must make sure it compiles and runs correctly on Ubuntu Linux 22.04 with GCC version 11.2. The performance evaluation should be done on your own computer in a Linux environment.

Your sorting application could read a large file and sort it in place (your program should be able to sort larger than memory files, also known as external sort). You must generate your input data by gensort, which you can find more information about at http://www.ordinal.com/gensort.html. You will need four datasets of different sizes: 1GB, 4GB, 16GB, and 64GB.

This assignment will be broken down into several parts, as outlined below:

Shared-Memory External Sort: Implement the Shared-Memory TeraSort application. You must use the following programming language (C) and abstraction (PThreads). Libraries such as STL or Boost cannot be used. You should make your Shared-Memory TeraSort multi-threaded to take advantage of multiple cores and SSD storage (which also requires multiple concurrent requests to achieve peak performance). You want to control concurrency separately between threads that read/write to/from disk, and threads that sort data once data was loaded in memory. Your sort should be flexible enough to handle different types of storage (where you need different number of threads), and different compute resources (where you need different number of threads based on the number of cores). You must implement your own I/O routines and sorting routines. If your system has more memory than your datasets to sort, you must limit the memory usage of your shared memory and Linux sort to 8GB. Note that in-memory sort might be faster than external sort (which has to be used for the 16GB and 64GB datasets. You need to implement a smart enough sort system that will detect the workload size and sort with the best approach possible (in memory for small datasets and external for larger datasets). You should mimic the command line arguments of the Linux sort program for your own shared memory sort benchmark, as much as possible.

Performance: Compare the performance of your shared-memory external sort with that from Linux "sort" (more information at http://man7.org/linux/man-pages/man1/sort.1.html) on a single node with all four datasets. You should vary the number of threads in your shared memory sort and figure out the best number of threads to use for each dataset size. The ideal number of threads might be different for your shared memory sort compared to the Linux sort (see --parallel=N command line argument to sort). Fill in the table below, and then derive new tables or figures (if needed) to explain the results. Your time should be reported in seconds, with an accuracy of milliseconds.

Complete Table 1 outlined below. Perform the experiments outlined above, and complete the following table: *Table 1: Performance evaluation of Single Node TeraSort (using best # of threads for each case)*

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								
Sort Approach (e.g. in-memory / external)								
Sort Algorithm (e.g. quicksort / mergesort / etc)								
Data Read (GB)								
Data Write (GB)								
Sort Time (sec)								
Overall I/O Throughput (MB/sec)								
Overall CPU Utilization (%)								

Average Memory Utilization (GB)								

For the 64GB workload, monitor the disk I/O speed (in MB/sec), memory utilization (GB), and processor utilization (%) as a function of time, and generate a plot for the entire experiment. Here is an example of a plot that has cpu utilization and memory utilization (https://i.stack.imgur.com/dmYAB.png), plot a similar looking graph but with the disk I/O data as well as a 3rd line. Do this for both shared memory benchmark (your code) and for the Linux Sort. You might find some online info useful on how to monitor this type of information (https://unix.stackexchange.com/questions/554/how-to-monitor-cpu-memory-usage-of-a-single-process).
After you have both graphs, discuss the differences you see, which might explain the difference in performance you get between the two implementations. Make sure your data is not cached in the OS memory before you run your experiments.

3. What you will submit

The grading will be done according to the rubric below:

- Shared memory sort implementation/scripts: 50 points
- README.md: 5 points
- Performance evaluation, data, explanations, etc: 40 points
- Followed instructions on deliverables: 5 points

The maximum score that will be allowed is 100 points.

You must have working code that compiles and runs on Ubuntu Linux 22.04 to receive credit for report/performance. A separate (typed) design document (named hw5-report.pdf) of approximately 1-3 pages describing the overall benchmark design, and design tradeoffs considered and made. Add a brief description of the problem, methodology, and runtime environment settings. You are to fill in the table on the previous page. Please explain your results, and explain the difference in performance? Include logs from your application as well as valsort (e.g., standard output) that clearly shows the completion of the sort invocations with clear timing information and experiment details; include separate logs for shared memory sort and Linux sort, for each dataset. Valsort can be found as part of the gensort suite (http://www.ordinal.com/gensort.html), and it is used to validate the sort. As part of your submission, you need to upload to your private git repository a build script, the source code for your implementation, benchmark scripts, the report, a readme file, and 6 log files.

A detailed manual describing how the program works (README.md). The manual should be able to instruct users other than the developer to run the program step by step. The manual should contain example commands to invoke the benchmark. This should be included as README.md in the source code folder.

Here are the naming conventions for the required files:

- Makefile
- mysort.c
- hw5_report.pdf
- README.md

- mysort1GB.log
- mysort4GB.log
- mysort16GB.log
- mysort64GB.log

- linsort1GB.log
- linsort4GB.log
- linsort16GB.log
- linsort64GB.log

When you have finished implementing the complete assignment as described above, you
should submit your solution to the blackboard. The timestamp on the BB submission will be used to determine if the submission is on time. Please put all your homework content into one .zip file and upload it to the

blackboard.

The name of .zip should follow this format: "***Section_LastName1_FirstName1-Section_LastName2_FirstName2.HW1.zip***"

Submit step #1: put all your files and documents for submission in a zip file.
Submit step #2: Name the file correctly and submit it to the blackboard.

Grades for late programs will be lowered 10% per day late.