COL730 Assignment 4: Sorting with MPI + OpenMP

Problem statement:

Design and implement two sorting algorithms (namely *Sort₁* and *Sort₂*) both targeting hybrid memory (shared and distributed) architecture using OpenMP and MPI frameworks. *Sort₁* should be an extension of the distributed algorithm designed in Assignment 3 (extended to exploit both shared and distributed memory models). *Sort₂* should be a completely different hybrid algorithm and should be stable i.e., the relative ordering of any two records with equal keys is preserved.

The dataset consists of key-value pairs of 16 bytes each. Each record consists of a 4-byte unsigned integer key and the value is a 12-character string. The algorithms are expected to sort the dataset by their keys while preserving the contents of the values corresponding to each key.

The sorting procedure is split into 3 collective functions:

a. *read*() takes a filename containing the entire dataset, decides how the dataset is divided between processes and for each process, reads its share of the dataset into memory. It returns a pointer to the local dataset and number of records in it. Each process calls the function with the same filename.

b. *sort*() takes a local dataset and one of two sorters, describing which algorithm to use. It modifies the input buffer so that the records are sorted by keys (both locally and globally with respect to the process rank). Note that the function may not alter number of local records. Each process calls the function with the same sorter.

c. *write*() takes a local dataset and a filename, writes the entire dataset into the file such that each local dataset is written contiguously and for each process, its local dataset comes before the datasets of processes ranked higher. Each process calls the function with the same filename.

The driver program initially calls read() to get the in-memory representation of the dataset. Then sort() is called on the dataset to receive the sorted dataset. Finally write() is called on the sorted dataset. Combining the specifications of sort() and write(), calling write() on a sorted dataset should result in a file containing sorted records.

MPI I/O functions should be used for reading/writing to/from files. A writer program "gen.cpp" is provided that creates a file containing a sample dataset. Note that the program is essentially sequential and is provided only to clarify the file representation of the dataset. The same format should be assumed in functions *read*() & *write*().

Total size of the dataset is at most 1/3rd of the combined memory of all the nodes being utilized. In other words, if equally distributed, local datasets will take up no more than 1/3rd the memory of a single node.

The algorithms are expected to scale to 16 processes (1 process per node) and 20 threads per node with a maximum input size of 2^{32} records.

Submission Instruction:

Submit a single zip file named [Your Entry Number].zip on moodle with the following:

1. An outline of the designed algorithms, the choices made and why.
2. Tables of execution times corresponding to environment configuration.
3. Sources implementing the function signatures provided in header “psort.h”.
4. A makefile that builds a library named “psort” (libpsort.a or libpsort.so).

Check course webpage for last date of submission.

Note:

Performance will be measured for all 3 functions. The sorting algorithms will be given a higher weightage than read/write.

The algorithms are expected to adapt dynamically to the number of available resources in the environment on execution. Such parameters should not be hardcoded into the implementation.

All global OpenMP & MPI configurations should be performed by init() and reverted by close() (if any).

You are expected to write your own driver program for testing and documentation for different configurations as necessary, but it may not be included in your submission.