1 Design for Assignment3

COL730 Assignment 3

Ashwini Kumar

2018MT60778

In this assignment, we have implemented merge sort and quick sort using a combined version of OpenMP and OpenMPI. Let us explain them.

2 Quick Sort:

As we did in the OpenMPI version,first we take up the last processor and then choose the pivot value for all processors to be the mid element of the 0th processor. The rank 0 processor broadcasts it to every processor. Then we take a partner process for each process to be the process at rank=(number_of_processes-my_rank). Then keep the data less than or equal to (in both the current process and the partner process) in the current process if it belongs to the lower half and send the data elements greater than pivot to the partner process which belongs to the upper half of the list of processors. Then split the communicator into two halves and then call recursively on each of the communicators.

This communication is continued till the point there is only one processor in the communicator. When there is only one processor, then we use OpenMP implementation or the shared memory implementation to carry out further computation i.e. when we have only one processor ,then use multithreading to sort the data elements.
For OpenMP/Shared memory implementation, first we decide a pivot to be the value as the last element of the array and then calculate the partition index as we do in sequential version of quick sort. The array is then separated into two portions- left and right array. We then use two sections in a recursive call of quick sort and assign left half to one of them and the other half to other section. This is continued recursively till we reach a threshold data size where we switch to sequential quick sort to remove the overhead created by parallelization on a small data set.

This way we sort the given data set using Combined Quick sort(OpenMP +OpenMPI).

2.1 Some important points:

The significant differences from the algorithm mentioned in the term paper and my implementation is that
(1) In OpenMP implementation, I have used sequential quick sort when data size becomes equal to some certain data size whereas the algorithm mentioned in the term paper uses number of threads as the criteria to decide when to switch to sequential version.
(2) I have splitted the communicator into two communicator instead of using groups as we saw in the term paper.

2.2 Observations:

OpenMP performed the best on the same data size as compared to Combined(Hybrid) version and then OpenMPI. i.e. OpenMP/Shared Memory Model> Combined(Hybrid) Model > OpenMPI/Distributed Memory Model.

3 Merge Sort:

We start by having the data on each processor so then we choose a helper process on which we send one half of the data elements. This processor then sends the half of this data sent, to another helper process, in a recursive call. This is carried out until every process has at least once received the data elements from some parent process. Thus each process now has some data elements for which we use OpenMP/Shared memory implementation of merge sort.

For OpenMP implementation, we split the given array into two halves. We then similarly use sections and assign each half of the array to each one of the sections. These sections call merge sort recursively on each half of the array. This recursive call is made till we reach a certain data size after which we switch to sequential merge sort. After each half is sorted, we merge these two halves as we do in sequential merge sort.

This way data elements are sorted using Combined Merge Sort(OpenMP +OpenMPI).

3.1 Some important points:

The significant differences from the algorithm mentioned in the term paper and my implementation is that
(1) In OpenMP implementation, I have used sequential merge when data size becomes equal to some certain data size whereas the algorithm mentioned in the term paper uses number of threads as the criteria to decide when to switch to sequential version.
(2) In term paper we assume data is given at one process but here we have data distributed on each process and then we sort them such that ith processor contains elements less than elements at the jth processor for i<j.

3.2 Observations:

OpenMP performed the best on the same data size as compared to Combined(Hybrid) version and then OpenMPI. i.e. OpenMP/Shared Memory Model> Combined(Hybrid) Model > OpenMPI/Distributed Memory Model.

Data Size	**Merge Sort**	**Quick Sort**
2^{10}	0.00572	0.000809
2^{15}	0.01205	0.012691
2^{18}	0.2754	0.317935
2^{20}	0.93	0.674325
2^{22}	1.50547	1.393
2^{24}	4.6574	3.727
2^{26}	24.983	15.0589

Table 1: Combined(Hybrid) Shared and Distributed Memory Performance Analysis