Myat Mo - 5417807

CSCI 5421

April 2, 2020

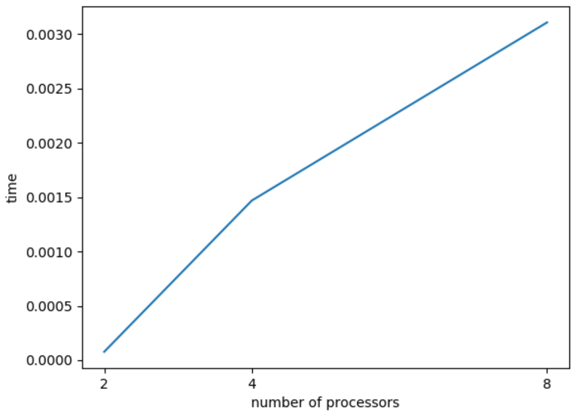
Lab 2 Solution

1. In main code, MPI_Barrier is used to wait until all processes in the communicator have reached that point so that it is safe to use all buffers passed to them after they return. In the main code, MPI_Bcast is used to broadcast list of pivots to all processors from the root. MPI_Gather and MPI_Gatherv (gatherv allows variable length of data) are used to collect data from all processors to the root.

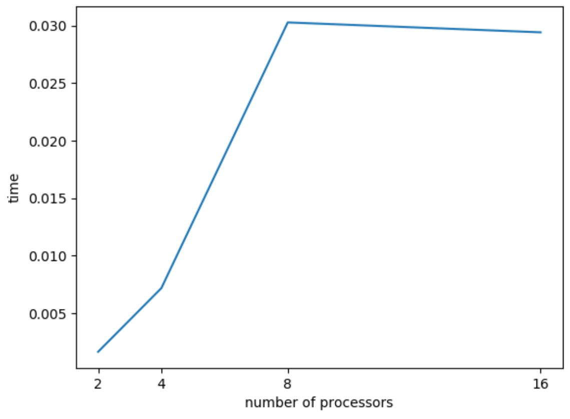
   In my function, I tried using both blocking and non-blocking Send. Sends and Receives commands are used to send/receive the length of the message first, then send/receive the message. Other than Send and Receives commands, MPI_Barrier is used to wait for all the process after each j loop. After the data is sorted in each processor in the last j loop, each processor now has different lengths. MPI_Gather is used to collect all the new lengths from all processor in the root processor. Then, MPI_Gatherv is used to collect different length of data from all processors in the root. After that, MPI_Scatter is used to distribute equal length of n to all processors back.

2. nprocs = [2, 4, 8, 16]
   For N=400:
   runtime = [7.700920e-05, 1.468897e-03, 3.108025e-03]
   For N=100,000:
   runtime = [1.655102e-03, 7.185936e-03, 3.028607e-02, 2.942514e-02]

   In each runtime, the first value is corresponded to the first value of nprocs. For eg, the first runtime value is for 2 processors and the second runtime value is for 4 processors etc.



(a) run-time analysis for total N=400



(b) run-time analysis for total N=100,000

3. As we see in the graphs above, the time is increases as the number of processors increases. This is due to the communication time betwee the processors. As the number of processors increases, the communication time between processors increases as well. However, if we compare the time of the same number of processor for N=400 and N=100,000 we can see that the time taken is reduced as N gets larger. Thus, to maintain the same level of efficiency and speed-up as the number of processors increases, the total number of data N must be increased.

   Hierarchical mapping of a task-dependency graph is used to distribute pivots to n processors. Each node represented is a supertask, and the partitioning of the arrays represents subtasks, which are mapped onto n processes. For example, the pivots are stored as a binary tree structure and the first pivot is mapped to all processors. On the second level of tree, pivot[1] and pivot[2] are mapped to processor 0 to $n/2$ and processor $n/2+1$ to n respectively. As we go down the tree, the number of processors is halved for each pivot. This prevents the imbalanced tasks among processors. However, choice of pivots can result load imbalance while sorting the data in each processor.

4. The pivot selection is not good when there are small number of data in each processor. Sometimes, load-imbalance occurs when all the data in one processor is smaller than the pivot. Then that processor is trying to send none of the data which result in MPI_ERR_COUNT. For now, the program will exit with an error message.

   Instead of using medians as pivots, we can use randomized partitioning in which pivot elements are selected randomly from the subcubes. That way, we can avoid the worst-case split and achieve better running time. And, even we add a few new levels with the most unbalanced split possible between these levels, the total time remains the same.