MPI Project 01

Goal: The goal of this project is to practice basic MPI routines (point-to-point communication in particular) and C++ STL by developing a MPI C++ code for sorting integers in parallel. It generalizes the basic sorting.cpp code we discussed in the class by combining it with idea explained in the parallel summation code.

Instructions

The base code to start is `~zxu2/Public/MPI_project01/sorting.cpp`.

1. To setup MPI environment, use command:

   ```
   module load mpich/3.3/intel/19.0
   ```

2. To compile the code, use command: `mpicxx sorting.cpp -std=c++11`

3. To run the program with 3 processes, use command (ideally, a script should be used):

   ```
   mpiexec -np 3 ./a.out
   ```

The parallel algorithm of the code to be developed is as follows:

Let's say we run this program with "total" number of processes. The size of the integer array is "N", which is defined as a macro in the code.

A. Process "0" uses random number generator to generate "N" integers and store them in `int_to_sort[]` array. To do very large N, we can change `int_to_sort` to be a pointer and use dynamic memory.

B. Process "0" send subsets of ints from int_to_sort[] to all other processes.
   a) let "j" be the rank of a process who is to receive ints from process 0. We use "`startval = BLOCK_LOW(j,total,N);`" to compute the beginning index of the ints in int_to_sort[] to send to process "j". The size of the ints to send to process "j" is computed by "`BLOCK_SIZE(j,total,N)`".
   b) Process "0" call MPI_Send() to send ints to processes "1" to "total-1". Processes "1" to "total-1" call MPI_Recv() to receive ints from process "0", respectively.

C. All processes use C++ std::set<int> sort_set to sort these subsets of ints, respectively. And save the sorted ints to dynamic array new_arr, whose size is "toal_loc_value = sort_set.size()".

D. Process "0" combines these sorted "new_arr" as follows:

a) Since std::set<int> only saves unique values, the sizes of the sorted ints in "new_arr" on the processes might be different.

Processes with rank values from "1" to "total-1" call MPI_Send() to send these sizes "toal_loc_value" to process "0". Process "0" call MPI_Recv() correspondingly.

b) Process "0" uses the received "toal_loc_value"s to allocate

a set of arrays for saving sorted values to be received.

c) Processes "1" to "total-1" call MPI_Send() to send "new_arr" to process "0". Process "0" calls MPI_Recv() to receive.

d) Process "0" call merge() function several times to combine all received ints from step D(c).

Optional: change all blocking communication calls to nonblocking calls. (We can try this after discussing the nonblocking point-to-point communication)

To develop and test the code, use "#define N 10" and 3 MPI processes.

In this way, we can check the output files for correctness of the program by looking at the results.

Next, change N to 100, 1000, 10000 and number of MPI processes to 3, 4, 5 respectively and use the job script to the program.

Timing computer times for each run. This is done by MPI_Wtime() function.

In the project report， please explain in detail how you implement step D.

It should contain: comments of variables used for this step; how you implement these communications; and computer times for each of runs.

Submission: please include source code and the project report.

Last remark:

The sorting algorithm implemented here is *by no means parallel efficient*. This is because the last merging step is purely sequential, only one MPI process is active. The goal of this project is to practice point-to-point communication and C++ features.