CX 4220/CSE 6220 Introduction to High Performance Computing
Spring 2023
Programming Assignment 2
Due Wednesday, March 29th, 11:59 PM

1 Problem Statement

Write a C/C++ parallel program to sort integers in parallel by doing a straightforward parallelization of the sequential quicksort algorithm. The algorithm is described here at a high level and you should fill in the remaining implementation details.

2 Parallel Algorithm

Let n denote the number of integers and p denote the number of processors. The integers are equally distributed to the processors: Each processor has an array A holding either $\lceil \frac{n}{p} \rceil$ or $\lfloor \frac{n}{p} \rfloor$ integers. The algorithm is recursive: At some stage during the algorithm, suppose there is a communicator of size q on which m integers should be sorted and that they are distributed in the usual manner. If $q = 1$, then any serial sorting algorithm can be used to sort the input: you either write this code, copy from a book or website if available, or even use the Unix sort program.

If $q > 1$, all processors in the communicator use the same random number generator to generate a random number between 0 and $m - 1$, say k. The processor that has the k^{th} integer (called pivot from here on) broadcasts it to all processors in the communicator. Each processor goes through its integer array and partitions it into two subarrays − one containing integers less than or equal to the pivot and another containing integers greater than the pivot. Using an *Allgather* operation, the number of integers in each of the two subarrays on all the processors are gathered on every processor. Compute the total number of integers less than or equal to the pivot (say m') and the total number of integers greater than the pivot (say m''). Partition the p processors to the two subproblems of sorting m' integers and sorting m'' integers respectively, by allocating processors in proportion to the problem sizes (make sure you do not allocate zero processors to a problem). Using the gathered information on the sizes of the subarrays on all processors, each processor can compute where to send its data and from where to receive its data. Create two new communicators corresponding to the two partitions, and use an *Alltoall* communication to perform the data transfer. Recursively sort within each partition.

3 Code framework

3.1 Input & Output Format

The input to the algorithm is a file containing the number of integers to be sorted followed by in the next line, the integers themselves (separated by space). The output of the program is the sorted list of space-separated integers in the first line and the time taken in ms (round to 6 decimal places) stored in the second line which should be written to an output file. The program should take 2 command line arguments, `input_file`, and `output_file`. You have to structure the program such that one processor (with rank 0) reads the input file and block distribute among all the processors. We are doing this for convenience sake, and because we do not have a machine that supports truly parallel I/O. Note that when you time your algorithm, you should start the timer after the data has been block distributed and end it before you write the sorted array to the output file. Sample input and output files are provided for reference in the `Programming Assignments` folder on Canvas. Sample command line input:

```
$ mpiexec -np 4 ./pqsort input.txt output.txt
```

3.2 Deliverables

The programming assignment is to be done in groups of three. It is important that you strictly adhere to the input format described in the programming assignment. No matter how you decide to share the work amongst yourselves, each student is expected to have full knowledge of the submitted program. To submit the programming assignment, turn in the following:

1. Create a Makefile for your program, and make sure the name of your output executable is "`pqsort`". If you are not familiar with creating Makefiles, check the resources below for help.

2. Experimentally evaluate the performance of your program. For instance, fix a problem size, vary the number of processors, and plot the run-time as a function of the number of processors. Fix the number of processors and see what happens as the problem size is varied. Try to come up with some important observations on your program. What is the minimum problem size per processor to get good parallel efficiency? We are not asking that specific questions be answered but you are expected to think on your own, experiment and show the conclusions along with the evidence that led you to reach the conclusions. Include your plots and observations in a PDF file with name "`report.pdf`". **Make sure to list names of all your teammates at the very beginning of your report. Also, provide a brief description of your implementation along with space and run-time analysis, and include a table at the end containing the contributions of each team member.**

3. Submit (*a*) all the source files and the Makefile in "Programming Assignment 2 Code" on Gradescope. Make sure you are not uploading the executable file in your submission, and (*b*) submit your report in "Programming Assignment 2 Report" on Gradescope. **Only one student needs to make the submission for the whole group. If students of the**

same group makes two different submissions, there would be a minor penalty applied.

4 Grading (100 pts total)

The program will be graded on correctness and usage of the required parallel programming features. You should also use good programming style and comment your program so that it is understandable. Grading consists of the following components:

1. Correctness (60 pts)
   - Automated tests on Gradescope that will test your program functionality for various inputs including all corner cases.
2. Performance (20 pts)
   - We will go through your code to make sure appropriate parallel programming practices discussed in the class are being followed along with the right MPI functions being called. Particularly, we will check the implementations for:
     - Dividing processes into sub-communicators after split
     - Load distribution within a communicator
     - Efficient data transfer among processes
     - Correct parallel recursive calls

     `Note`: Points will be deducted for ignoring performance protocols; **serialization in the program will lead to a zero on the whole assignment.**
3. Report (20 pts)
   - Two plots (10 pts), theoretical analysis for space and time complexities (3 pts), empirical analysis, observations, and conclusion (7 pts).

5 Resources

1. What is a Makefile and how does it work?: https://opensource.com/article/18/8/what-how-makefile
2. PACE ICE cluster guide: https://docs.pace.gatech.edu/ice_cluster/ice-guide/. Documentation for writing a PBS script: https://docs.pace.gatech.edu/software/PBS_script_guide/
3. Sequential quicksort algorithm:
   https://www.cs.princeton.edu/ wayne/kleinberg-tardos/pdf/05DivideAndConquerI.pdf ‘