Assignment no.3
The Parallel Quicksort algorithm

Roman Iakymchuk, TDB, IT Dept
roman.iakymchuk@it.uu.se

April 26, 2024

1 Problem setting

Given a sequence of n integers, the task is to implement the Parallel Quicksort algorithm using C and MPI, and evaluate the performance. You are encouraged to work in pairs. However, a statement of contribution with clear but brief description of who did what should be provided.

1.1 Outline of the algorithm

Algorithm 1.1 The Parallel Quick-sort algorithm

```
1   Divide the data into p equal parts, one per process
2   Sort the data locally for each process
3   Perform global sort
    3.1  Select pivot element within each process set
    3.2  Locally in each process, divide the data into two sets according to
         the pivot (smaller or larger)
    3.3  Split the processes into two groups and exchange data pairwise be-
         tween them so that all processes in one group get data less than the
         pivot and the others get data larger than the pivot.
    3.4  Merge the two sets of numbers in each process into one sorted list
4   Repeat 3.1 - 3.4 recursively for each half until each
    group consists of one single process.
```

The algorithm converges in $\log_2 p$ steps.

1.2 Pivot strategies

You are expected to implement the following pivot strategies:

1. Select the median in one processor in each group of processors.

2. Select the median of all medians in each processor group.
3. Select the mean value of all medians in each processor group.

You are free to test other pivot selecting techniques as well if you like.

2 Implementation

Your program is supposed to take three arguments.

1. The first argument is the name of a file containing data to be sorted (the input file).
2. The second argument is the name of the file to which the program will write the sequence of sorted numbers (the output file, in ascii format, as the input file).
3. The third argument is a number in the range $1 - 3$ specifying which pivot strategy to use, according to the list in Section 1.2. If you choose to implement other pivot strategies, you may of course extend the range.

The input file will contain $n + 1$ numbers. The first number is n (that is, the number of elements to sort) and the n subsequent numbers are the actual number sequence. The output file shall contain these numbers, sorted, and nothing else. The numbers should be separated by white space. This holds also for the input file.
Your implementation must be independent of n. Assume that the number of processes is equal to 2^k for some non-negative integer k.
For Step 2 in Algorithm 1.1 you may use the C function `qsort`. You are supposed to sort the numbers in ascending order.
Measure the time for sorting in seconds, not including file reading and writing. The number of seconds (and nothing else) shall be written to stdout when the sorting is done.
Note that your code should be reasonably well structured and documented.

3 Performance experiments

It is recommended to run the numerical experiments on Snowy at UPPMAX. Vary the number of processing elements and compute the corresponding speedup. Perform a number of experiments to demonstrate strong and weak scalability of your implementation. Run your program on sequences of n random numbers with different values of n. Also, evaluate the performance for a sequence of numbers sorted in descending order. Compare the different pivot strategies for both types of sequences.
When planning: You may use the files stored in the directory `/proj/uppmax2024-2-9/nobackup/A3` at UPPMAX as input data. Larger files are intended for numerical experiments, while smaller files can be used for testing of your program.
Files whose name starts with 'input' contain sequences of random numbers and files whose name start with 'backwards' contain elements sorted in *descending* order. The number in

the file name is the size of (i.e. the number of elements in) the sequence contained in the file. All files follow the format described in Section 2.
Note that you have a limited disc quota at UPPMAX! Therefore, it is recommended that you don't copy the larger input files to your home directory, but read them from the project directory (where they are stored now). Moreover, *it is a good idea to remove large output files as soon as possible.* You may also skip the printing to files during the numerical experiments. However, note that the code that you hand in shall follow the specifications above.

4 Precaution

Use the UPPMAX resources with care! Read once again the 'running jobs on clusters' announcement on Studium and follow Usama's suggestions given on Monday, April 22nd. We are given 2000 core/hours for the whole course.

5 Report

You are supposed to write a report on your results. The report should contain:

1. A brief description of the theoretical problem and your implementation.
2. A description of your numerical experiments.
3. The results from the performance experiments. Execution times for different problem sizes and different number of processing elements should be presented in tables. Speedup values should be presented both in a table and in a plot. Also plot the ideal speedup in the same figure. The plots can be done using MATLAB, or any other tool that you are familiar with.
4. A discussion of the results. Does the performance of the parallel program follow your expectations? Why/why not?

6 File to be submitted

The file that you upload in Studium must be a compressed tar file named A3.tar.gz. It shall contain a directory named A3, with the following files inside:

1. A PDF file named A3_Report.pdf containing you report.
2. Your code, following the specifications listed in Section 2. Note that the code must compile without warnings on the Linux system!
3. A Makefile that does the following.

(a) Builds a binary named `quicksort` from your code when the command `make` is invoked. This must work on the Linux system as well as on UPPMAX!

(b) Removes all binary files and object files when the command `make clean` is invoked.

Since your code will be tested automatically, it is very important that you follow the instructions regarding the implementation and the structure of the tar file! Failure to do so will result in immediate rejection of the assignment.
Before submitting, please check your file using the script `check_A3.sh` included in `A3.tar.gz` (which you have downloaded from Studium):

1. Put `check_A3.sh` and `A3.tar.gz` in the same directory at Snowy. Go to that directory
2. Make sure that `gcc` and `openmpi` are loaded: `module load gcc openmpi`
3. Submit the script to the batch system: `sbatch ./check_A3.sh`
4. Check the output of the script as you learned in Lab 2.

If the output says that your file is ready for submission, you may submit it. However, note that this is not a guarantee that you will pass the assignment. The teachers will perform additional tests and also consider your report when grading. If you do not get a message that your file is ready for submission, please fix it according to the output of the script. If your program has produced the wrong result, you can find the result in the file `output_to_remove.txt` in the directory `A3`.

Important note: When you upload a first revised version of your work, change the name of the file as `A2-R1.tar.gz` and if a second revision is allowed - `A2-R2.tar.gz`.

Best wishes,
Roman Iakymchuk

Any comments on the assignment will be highly appreciated and will be considered for further improvements. Thank you!