CSCI 3366 (Parallel and Distributed Programming), Fall 2019

Homework 5

Credit: 20 points.

1 Reading

Be sure you have read, or at least skimmed, chapters 1 through 5 of the textbook.

2 Overview

Your main mission for this assignment is write a parallel version of the well-known sorting algorithm quicksort, in Java. (If you don't remember the details of quicksort, the Wikipedia article seems reasonable. The algorithm I want you to implement is the one that sorts in place. I also like the discussion here.)

3 Details

3.1 Sequential program

(5 points)

To help you get started, I wrote a starter program to test and time the sort:

- Code: QuickSortSeq.java. (Note that the class this defines is in package `csci3366.hw4`, so it should go in a directory named `csci3366/hw4`.)

Comments in the code explain the command-line parameters. You will need to fill in the body of the `sort` method. Feel free to find code on the Web or in a textbook (though you should try to understand the code you use even if you don't write it yourself), and/or to add additional methods. (If you've never personally written an implementation of the quicksort algorithm, this would be a good opportunity to do so; it's something everyone who uses this algorithm should probably do at some point!)

3.2 Parallel program

(10 points)

The next step is to write a parallel version of your program that allows you to specify the number of threads with an additional command-line argument. Of course(?), the program should print the number of threads along with timing results. You can use the mergesort example from the "sample programs" page as a model; it's a reasonable choice for a first implementation. Note however that, unlike mergesort, quicksort doesn't necessarily split the array into two pieces of roughly equal size, so you may not get very good performance for this approach. Still, it's a good first try and for this assignment will be good enough for full credit. I'll give extra credit (up to 5 points) if you do something more sophisticated.

3.3 Performance of parallel program

(5 points)

Once you have working parallel code, experiment with input values until you get a problem size/configuration big enough to make it reasonable to hope for good speedups with multiple UEs. (Given how quicksort works, once you find a problem size that seems big enough, you may find it interesting to try out more than one seed.) Then time your parallel program for at least two inputs (combination of problem size and seed) and different numbers of UEs and plot the results, as in Homeworks 3 and 4.

4 What to turn in and how

Turn in the following:

- Source code for your sequential and parallel program.
- Results of measuring performance. For each of these programs, tell me what inputs you used for the program, which machine(s) you ran it on, and send me:
  - A plot showing how execution time depends on number of UEs.
  - Input data for the plot. A text file or files is fine for this.

Submit your program source code by sending mail to `bmassing@cs.trinity.edu`. Send program source as attachments. You can turn in your plots and input data as hardcopy or by e-mail; I have a slight preference for e-mail and a definite preference for something easily readable on one of our Linux machines — so, PDF or PNG or the like (in the past I think some students have sent me Excel spreadsheets, which — I'd rather you didn't). Please use a subject line that mentions the course number and the assignment (e.g., "csci 3366 homework 5").

5 Honor Code Statement

Include the Honor Code pledge or just the word "pledged", plus *at least one of the following* about collaboration and help (as many as apply).[1] Text *in italics* is explanatory or something for you to fill in. For programming assignments, this should go in the body of the e-mail or in a plain-text file `honor-code.txt` (no word-processor files please).

- This assignment is entirely my own work. *(Here, "entirely my own work" means that it's your own work except for anything you got from the assignment itself — some programming assignments include "starter code", for example — or from the course Web site. In particular, for programming assignments you can copy freely from anything on the "sample programs page".)*
- I worked with *names of other students* on this assignment.
- I got help with this assignment from *source of help — ACM tutoring, another student in the course, the instructor, etc. (Here, "help" means significant help, beyond a little assistance with tools or compiler errors.)*

[1] Credit where credit is due: I based the wording of this list on a posting to a SIGCSE mailing list. SIGCSE is the ACM's Special Interest Group on CS Education.

- I got help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. (Here too, you only need to mention significant help — you don't need to tell me that you looked up an error message on the Web, but if you found an algorithm or a code sketch, tell me about that.)*
- I provided help to *names of students* on this assignment. *(And here too, you only need to tell me about significant help.)*

6 Essay

Include a brief essay (a sentence or two is fine, though you can write as much as you like) telling me what about the assignment you found interesting, difficult, or otherwise noteworthy. For programming assignments, it should go in the body of the e-mail or in a plain-text file `essay.txt` (no word-processor files please).