CSC2002S Tutorial 2021

Parallel Programming with the Java Fork/Join framework: 1D Median Filter

Median filtering is a nonlinear digital filtering technique that is often used to remove noise from a data set. Noise reduction is usually a pre-processing step to that improves or "cleans" the data in preparation for later processing. In this method, a median filter slides over the data item by item, generating a new data set, where each item is replaced by the median of neighbouring entries. An example result for a 1D data set is shown in figure 1 below.

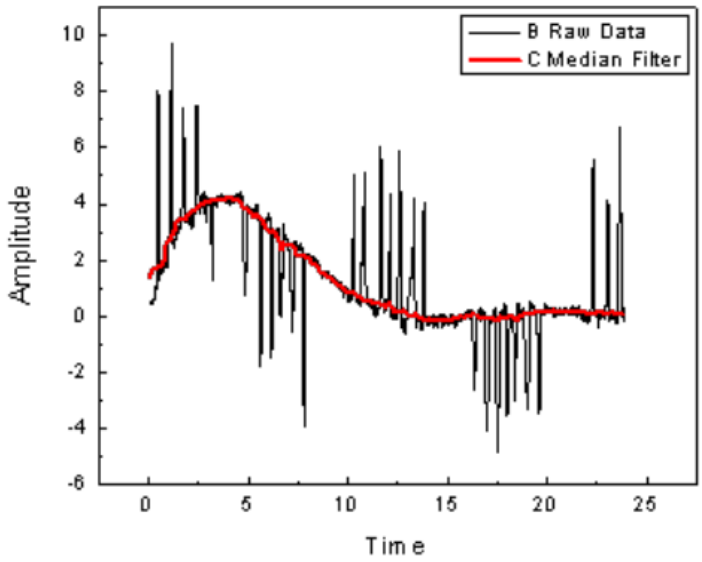

Figure 1: An example of median filtered data.

The size of the filter determines the number of neighbours considered for the median. In this assignment, you will consider only one-dimensional data sets and filters of odd size, ranging from 3 to 21 data items.

For example, a median filter of size 3 applied to the simple 1D array x:

x = [2 80 6 3 1]

produces output array y:

y = [2 6 6 3 1]

where each element of y is calculated as follows:

y[1] = Median[2 80 6] = 6
y[2] = Median[80 6 3] = 6
y[3] = Median[6 3 1] = 3

Note that the borders are not changed and that the size of the borders depends on the filter width. In this example, the border is 1 element wide. The naïve approach to median filtering sorts the elements within the filter window at each step to calculate the median (which will then be the middle element). Using this naïve approach (which we will do in this assignment) the median computation can be quite expensive, especially if the filter window is large. Therefore, in this assignment you will attempt to parallelize the problem in order to speed it up. You will:

- Use the Java Fork/Join framework to parallelize the median filter operation using a divide-and-conquer algorithm. Do not use any other method of parallelizing this problem,

as mentioned in lectures.

- Evaluate your program experimentally:
    - Using high-precision timing to evaluate the run times of your serial and your parallel method, across a range of input sizes and a range of filter sizes, and
    - Experimenting with different parameters to establish the limits at which sequential processing should begin.

- Write a report that lists and explains your findings.

Note that parallel programs need to be both **correct** and **faster** than the serial versions. Therefore, you need to demonstrate both correctness and speedup for your assignment. If speedup is not achieved, you need to explain why.

Assignment details and requirements

Your program will smooth one-dimensional time series data using a median filter. The filter will not smooth the endpoints.

Input and Output

Your program must take the following command-line parameters:
<data file name> <filter size (must be an odd integer >= 3)> <output file name>

You can assume that the data file starts with an integer on a single line, specifying the number of subsequent lines in the file. Successive lines contain two numbers separated by spaces:
<line number> <floating point value $0 < x < 5$ >

We will provide you with sample input files (on Vula), though you may also create your own using random data.

Your program should output the cleaned 1D data set in the same format as the input file, with an accuracy up to 5 decimal places.

1.2 Benchmarking

You must time the execution of your parallel classification across a range of data **sizes** and **number of threads** (sequential cutoffs) and report the speedup relative to a serial implementation. The code should be tested on a single **multi-core machine architecture.**

Timing should be done at least 20 times. Use the `System.nanoTime()` or `System.currentTimeMillis` methods to do the measurements. Timing must be restricted to the median filter operation. Timing should exclude the time to read in the file beforehand, and print the cleaned data afterwards, as well as other unnecessary operations. Call `System.gc()` to minimize the likelihood that the garbage collector will run during your execution (but do not call this within your timing block!).

1.3 Report

You must submit an assignment report **in pdf format**. Your clear and **concise** report should contain the following:

- An *Introduction*, comprising a short description of the aim of the project, the parallel algorithms, and their expected speedup/performance.
- A *Methods* section, giving a description of your approach to the solution, with details on the parallelization. This section must explain how you validated your algorithm (showed that it was correct), as well as how you timed your algorithms with different input, how you measured speedup, the machine architecture you tested the code on, and interesting problems or difficulties you encountered.
- A *Results and Discussion* section, demonstrating the effect of data sizes and numbers of threads and different architectures on parallel speedup. This section should **include speedup graphs** and **a discussion**. Graphs should be clear and labelled (title and axes). In the discussion, we expect you to address the following questions:
    - Is it worth using parallelization (multithreading) to tackle this problem in Java?
    - For what range of data set sizes and filter sizes does your parallel program perform well?
    - What is the maximum speedup obtainable with your parallel approach? How close is this speedup to the ideal expected?
    - What is an optimal sequential cutoff for this problem? (Note that the optimal sequential cutoff can vary based on dataset size.)
    - What is the optimal number of threads on your architecture?
- A *Conclusions* (note the plural) section listing the conclusions that you have drawn from this project. What do your results tell you and how significant or reliable are they?

Please do NOT ask the lecturer for the recommended numbers of pages for this report. Say what you need to say: no more, no less.

1.4 Assignment submission requirements

- You will need to submit a GIT usage log as a .txt file (use `git log –all` to display all commits and save this as a separate file).
- Your submission archive must consist of BOTH a technical report and your solution code and a **Makefile** for compilation.
- **Label** your assignment archive with your **student number** and the **assignment number, e.g., KTTMIC004_CSC2002S_Assignment1.**
- Upload the file and **then check that it is uploaded.** It is your responsibility to check that the uploaded file is correct, as mistakes cannot be corrected after the due date.

Target Deadline:

23:55 on 23rd August 2021

- Submissions will be accepted until the **hard deadline of 23:55 28th August 2021** with the standard penalty of 10% of the total mark per 24 hours or part thereof late submission. Submissions after the hard deadline will not be accepted.
- The deadline for marking **queries** on your assignment is **one week after the return of your mark.** After this time, you may not query your mark.

1.5 Extensions to the assignment

If you finish your assignment with time to spare, you can attempt one of the following for extra credit:

- Compare the performance of the Fork/Join library with standard threads.
- Compare the performance with a mean filter, which replaces a value with the mean of the surrounding elements (instead of the median).
- Implement a 2D dataset and a 2D median filter.
- Investigate and implement linear time solutions to median filtering.

1.6 Assignment marking

Your mark will be based primarily on your analysis and investigation, as detailed in the report.

Rough/General Rubric for Marking of Assignment 1	
Item	Marks
Code – conforms to specification, code correctness, timing, style and comments, produces correct outputs	30
Documentation - all aspects required in the assignment brief are covered, e.g. Introduction, Methods, Results (with graphs and analysis), Conclusions.	50
Going the extra mile - excellence/thoroughness/extra work - e.g. additional investigations, depth of analysis, etc.	10
GIT usage log	5
Makefile – compile, docs, clean targets, readme	5
Total	100

Note: submitted code that does not run or does not pass standard test cases will result in a mark of zero. **Any plagiarism, academic dishonesty, or falsifying of results reported will get a mark of 0 and be submitted to the university court**.