**CSE 332 Project 3 Write up**

\* Note: The last 3 questions require you to write code, collect data, and produce graphs of your results together with relatively long answers. Do not wait until the last minute to start this write up!

1. **Who is in your group (Your name, UW NetID)?**

Reggie Jones 1027248

Tristan Riddell …

1. **What assistance did you receive on this project? Include anyone or anything *except* your partner, the course staff, and the printed textbook.**

Java Docs

Stack overflow

Fork Join Framework notes by Dan Grossman

1. **a) How long did the project take?**

-Reggie: : ~45 phase A/B (coding/design). 10+ phase C

-Tristan:

**b) Which parts were most difficult?**

Refactoring the code was difficult. We didn’t exactly have the same vision for each version so it was hard to know what the other person was thinking and how they were trying to build it with optimal code reuse.

**c) How could the project be better?**

Giving more hints on how/what to test. Also less experimentation. I rather spend more time building cool stuff than experimenting. The coding time seems exponentially more beneficial in learning the material. I understand the experimentation is ment for analysis of the code to get a quantitative analysis, but it’s too daunting and MUCH LESS effective way of learning

1. **(OPTIONAL) What "above and beyond" projects did you implement? What was interesting or difficult about them? Describe in detail how you implemented them.**
2. **a) How did you test your program? What parts did you test in isolation and how?**

Reggie- I tested the BuildGrid, FindCorner, and FindPopulation classes. I tested both the sequential and parallel cases (the parallel cases by setting the sequential cutoff to a low number as the spec suggests). In order to test these 3 classes in isolation from one another I just created some small sample inputs that were necessary for each class to run.

I made 1 bash script that would run the PopulationQuery main for all 5 versions and compared the output to the output that Hye In gave us (I believe she posted it on the message board of some sample queries and outputs they should produce).

I made a bash script that ran the PopulationQuery main for all versions with the same random x-column and y-row arguments. Then several random queries would be run on all the versions (the same queries for each version). It then compares the output of all the versions together and does a diff to see if any of them are different from one another.

Also, I made a few simple bash scripts that would test malformed java arguments or population queries would notify the user the input was incorrect.

Tristan-

**c) What smaller inputs did you create so that you could check your answers?**

We created inputs off the top of our head and manually stepped through the code to figure out what the correct output should be. We then tested to check the input gave us the results we were expecting to see.

**d) What boundary cases did you consider?**

1. **For finding the corners of the United States and for the first grid-building step, you implemented parallel algorithms using Java's ForkJoin Framework. The code should have a sequential cut-off that can be varied. Perform experiments to determine the optimal value of this sequential cut-off.**

**1) Sequential vs. parallel versions of corner finding. Looking at V1 and V2, vary the cutoff for V2.**

**2) Cut-off in the grid-building step. Looking at V3 and V4, vary grid-building cutoff for V4.**

**3) Cut-off in the grid-merging step. Looking at V3 and V4, vary grid-merging cutoff for V4.**

\* You don't have to worry about finding the optimal combination of cut-offs.

**a) Describe your experimental setup:**

**1) Your machine characteristics**

Machine characteristics: With each new sequential cut off value, runs 1000 warm up tests followed by 5000 real tests. The time for that cut off is the average of the 5000 tests. The code simply calls ParallelCorners, our v1 and v2 parallel class. When sequential cut off is equal to data size, the code is strictly sequential.

**2) How you collected timing information**

I collected timing information using System.currentTimeMillis().

**3) Any details that would be needed to replicate your experiments**

No details needed to replicate experiment, merely rerun V2SequentialTest.java.

**b) Experimental Results: Place your graph for experiment 1), 2) and 3).**

Clearly label which line is for which version in each of your plot.

1) Cutoff vs. Runtime for V1 and V2

2) Cutoff vs. Runtime for V3 and V4: Grid building

3) Cutoff vs. Runtime for V3 and V4: Grid merging

**c) Interpretation of Experimental Results**

Note that if the sequential cut-off is high enough to eliminate all parallelism, then you should see performance close to the sequential algorithms, but evaluate this claim empirically (and then answer the question - is this what you see?). For each of the experiments 1), 2) and 3), answer the following questions.

**1) What did you expect about the result and why?**

**2) Did your result agree with your expectation?**

**3) If the result did not match with your expectation, why do you think it happened?**

**4) Draw a conclusion from the experimental result.**

I expected to see time efficiency peak with a lower cut off, around the 1000 to 10000 range. However, according to my data, time is nearly constant as long as there are at least two threads. For any cut off that calls multiple threads, the time is about ~1.3. But when the code is run FULLY sequential, the time is over 5 milliseconds which is almost four times as much as even 1000 less than the data size. This did not entirely match my expectations. Although I believe the reason it turned out this way is the data set is too small to really see a difference with a changing cut off. I imagine if we performed the same test on a data file with 100 million data points, we would find an optimal cut off much easier.

I believe that with a data set this small, the efficiency effectively maxes at 2 processors. Which means as long as there are two threads, the performance will be good. This would explain why time is constant for all cut off, with an enormous outlier when it was run fully sequentially.

1. **Compare the performance of V4 to V5 as the size of the grid changes.**

Clearly label which line is for V4 and which line is for V5 in your plot.

1. **Experimental Results: Place your graph for Grid size vs. Runtime for V4 and V5 here.**

**b) Intuitively, which version is better for small grids and which version is better for large grids?** Intuitively I would expect v4 to work better for small grids, and v5 to work for larger grids. The reason I think this is because if the grid is small than v4 will use many many threads and uses CombineGrids which is really fast for a small grid. I think itll be faster than v5 because that only have 4 threads total to parse the input array to build the grid. I think v5 will be faster for large grids because the opposite of the above- namely that CombineGrids Will be somewhat slow because it has to add large grids together. Also that v5 only has 1 large grid that it accesses concurrently which should give it an edge over v4 that has to replicate many and many and many of these large grids in order to add them together.

**c) Does the experimental data validate your hypothesis in b)? If the result did not match with your**

**expectation, why do you think it happened?**

Half of my hypothesis was correct- namely that v4 would be faster for a small grid (by an order of magnitude- hard to tell on the graph) . However,

It was also MUCH faster than v5 at large grid sizes. So much faster that it has made me question If there was a bad design choice or implementation detail. After spending quite a while digging I haven’t found out why. My best guess as to *why I think this happened* is something to do with our code. I simply do not think that v5 should be ~25 times slower for a 100x100 grid.

1. **Compare the performance of V1 to V3 and V2 to V4 as the number of queries changes. That is, how many queries are necessary before the pre-processing is worth it?**

**Clearly label which line is for which version in each of your plot.** **Note you should time the actual code answering the query, not including the time for entering the query.**

**a) Experimental Results (Place your graph here).**

1) Number of Query vs. Runtime for V1 and V3

2) Number of Query vs. Runtime for V2 and V4

**b) Interpretation of Experimental Results**

For each of the experiments 1) and 2), answer the following questions:

**1) What did you expect about the result and why?**

**2) Did your result agree with your expectation?**

**3) If the result did not match with your expectation, why do you think it happened?**

**c) According to your experiment, how many queries are necessary before the pre-processing**

**is worth it?**

1. **If you worked with a partner:**

**a) Describe the process you used for developing and testing your code. If you divided it, describe that. If you did everything together, describe the actual process used (eg. how long you talked about what, what order you wrote and tested, and how long it took).**

We didn’t have a clear cut process- we simply worked on it when we could and communicated to the other person “I’m doing version 3” or whatever it may have been. We divided most of it- Reggie did v1 v3 v4 and Tristan did v2 and v5. On v2 there was a lot of duplicate code so we worked together on this together in the lab for a good while. For testing it we divided it by pretty much testing the classes you wrote. We spent much more time collaborating back and forth on this project than project 2. It took a lot of hours! (specifics in question 3 above)

**b) Describe each group member's contributions/responsibilities in the project.**

Reggie: -v1

-refactoring v1 to work with GUI

-cmd line arg processing & user query input processing

-refactored v2 to share code instead of stand as separate entity of v1

-v3 & v4

-Tests (described which tests I did in the question above that asks about this)

Tristan: -

**c) Describe at least one good thing and one bad thing about the process of working together.**

One good thing is we are both pretty rad so collaborating/communicating was fairly easy. One bad thing is that this was hard to work on together. It seemed like much more of a sequential workflow than project 2 did. I think this was because the versions shared a lot of code so it was hard to work on it at the same time.

**Appendix**

Place anything else that you want to add here.