**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)?**

Yufang Sun ID: mandary

Boren Li ID: borenl93

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

Friend’s discussion, google, stack over flow. Eclipse is actually very helpful, it gives us a lot suggestion about how to organize the project and set the fields as protected or private.

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

Four days.

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

Understanding the spec and organizing the project. Also, getting together with partners who also have a busy schedule.

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

Wish the graphical interface was already implemented. It will be more interesting to run on different census files. Wish we can have more experience with locks and critical sections.

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?**

Junit test. We tested all user accessible function and partial internal functions. And compared with the standare output given. We also build our own small test file. We first isolated the finding corner step for precise population testing by comparing each version's result. We also tested illegal command line argument and user inputs.

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

We abridged the first 15 lines of data from CenPop2010.txt as test.txt for precise testing.

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

Whole population, or the whole rectangle. Place with 0 percent population, and we compare detailed output from all versions. We also tested the whole us map's four rectangle corners. We also tested all negative or illegal arguments and inputs.

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.**

For experiment control, we only test on CenPop2010.txt and all choose full population option. And the X: 100, Y: 500. We test each three times and calculate their average as final results, time unit as ms.

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

Optimal cutoff for V2 would be 50000 = about 210000/4 cores.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | V1 | V2:1000 | V2:5000 | V2:10000 | V2:50000 | V2: 100000 | V2: 250000 |
| First | 15 | 32 | 15 | 15 | 15 | 15 | 15 |
| Second | 16 | 31 | 30 | 32 | 15 | 15 | 16 |
| Third | 15 | 31 | 15 | 15 | 16 | 15 | 15 |
| Average: | 15 | 31 | 20 | 20 | 15 | 15 | 15 |

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

For experimental control: we use 10000 cutoff for V4 merging grid. V4 optimal cutoff: 50000

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | V3 | V4:1000 | V4:5000 | V4:10000 | V4:50000 | V4: 100000 | V4: 250000 |
| First | 0 | 126 | 47 | 31 | 15 | 16 | 15 |
| Second | 0 | 125 | 47 | 31 | 15 | 16 | 15 |
| Third | 0 | 124 | 47 | 31 | 15 | 16 | 15 |
| Average: | 0 | 125 | 47 | 31 | 15 | 16 | 15 |

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

For experimental control: we use 100000 cutoff for V4 building grid. V4 optimal cutoff: 100000

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | V3 | V4:1000 | V4:5000 | V4:10000 | V4:50000 | V4: 100000 |
| First | 0 | 16 | 15 | 16 | 16 | 15 |
| Second | 0 | 17 | 16 | 16 | 15 | 15 |
| Third | 0 | 17 | 16 | 15 | 15 | 15 |
| Average: | 0 | 16 | 15 | 15 | 15 | 15 |

**a) Describe your experimental setup:**

**1) Your machine characteristics**

CSE lab computer with Intel Core i7-3770 CPU @ 3.4 GHz, 8GB memory, Windows 7 Enterprise

**2) How you collected timing information**

By doing System current time millis for timing at where the described step starts and where it ends. We did have the difficulty to separate the grid building step and grid merging step in Experiment 2 and 3. But we hold an experimental control for the testing to happen. We varied the V4 cutoff of grid building method first, and then keep it constant at an optimal value to measure the grid merging step by varying the V4 cutoff of combining the two grids.

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

Note, there is a test class in our file, but we did not use it for our final testing.

You need to include the following statements at the desired step:

Experiment 1: V1, line 19 and 21. V2, line 22 and 24

Experiment 2: V3, line 21 and 23. V4, line 20 and 22, change the cutoff of building step in V4

Experiment 3: same as number 2 but change the cutoff of merging step in V4

At the beginning:

**final** **long** startTime = System.*currentTimeMillis*();

In the end:

**final** **long** endTime = System.*currentTimeMillis*();

System.*out*.println("Total execution time: " + (endTime - startTime) );

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

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?**

For all experiments, we expected the parallelism be faster because there’re more threads handling the same processes that sequential thread is handling by itself. Also, the sequential code is taking advantage of multi-core processors. We also believed that the processor we used can handle 8 threads parallel with 4 cores.

1. **Did your result agree with your expectation?**

Experiment 1: No. We did not see the expected result until cutoff is 50000

Experiment 2: No. We did not get expected result, sequential is faster! We are surprised that the sequential grid building step takes 0 ms.

Experiment 3: No. We did not get expected result, sequential is faster!

With the above experiment result, we can conclude that with this specific census data array with about 210000 number of data sets, sequential code K.O.ed parallel code badly.

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

In our opinion, the parallelism takes more time to initialize the pool and more threads taking turns for 4 cores aren’t efficient. Once parallelism exceeds four threads, we did not see optimal result compared to sequential code. We think more resources was spent on organizing the threads in pool. Also, because the data amount is not big at all, so we did not observe the benefit of parallelism.

Side note, we are also running word and internet browser at the same time, that might consume some cache, memory, or CPU usages and competing with the java framework.

1. **Draw a conclusion from the experimental result.**

From our experiments, parallel code with big enough cutoff isn’t necessarily the same as sequential code. Sequential code is actually faster! That might be caused by the extra initializing step of the ForkJoinPool and creating more threads. We know things don’t occurs simultaneously, so once the thread exceeds the number of cores, it became much slower. Also, the step of combining two grids, we used the special implementation of splitting the grids into four parts, leftup, leftdown, rightup, and rightdown. It may encounters the extreme case that two down cases have nothing to do and just hanging and waiting for the upper parts to be done. That said, implementation details and cutoffs are really important for parallel codes. In addition, we know things do take turns to get executed, the scheduling of the processes might also cause a big difference. In conclusion, parallelism for non-trivial and small amount of data isn’t worth the time parallelizing, because parallelism has a basic tradeoff initializing and handling the threads. Sequential code with small amount of data can give better performance due to nowadays hardware advancements and sequential optimization.

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

V4 uses 100000 for parallelizing census data and 5000 for combining the grids.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1000x, 500y** | **100x, 500y** | **100x, 250y** | **50x, 125y** | **20x, 25y** |
| **V4** | **47** | **15** | **16** | **15** | **0** |
| **V5** | **115** | **47** | **31** | **31** | **16** |

1. **Experimental Results: Place your graph for Grid size vs. Runtime for V4 and V5 here.**
2. **Intuitively, which version is better for small grids and which version is better for large grids?**

Version 4 is better for larger grids whereas Version 5 might be slower.

**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?**

It did match our hypothesis! ForkJoin program did beat up Lock-based threads pretty badly, because we think this is when the optimization of ForkJoinPool comes into place. Although we use sequential cutoff to split threads, but ForkJoinPool only create a certain amount of threads. Thus, ForkJoinPool is actually more efficient performance wise comparing the hardcode four threads out with Java Threads.

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.**

For experiment control, we use 100x and 500y, and the most optimal test result from the above experiments. All user input is “1 1 100 500”.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version | 1 | 5 | 10 | 20 | 30 |
| V1 | 0 | 16 | 16 | 30 | 60 |
| V3 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version | 1 | 5 | 10 | 20 | 30 |
| V2 | 0 | 0 | 15 | 31 | 60 |
| V4 | 0 | 0 | 0 | 0 | 0 |

**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**

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

We expected V3 and V4 be constant time answering the query because the way we implemented it. After building grid and updating its population, it should answer the query at constant time by just subtracting top left and add the diagonal corner. Unlike version 1 and 2, they spent O(n) to find the population each time.

1. **Did your result agree with your expectation?**

Yes.

**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?**

We think 20 times at least will worth the pre-processing from the run time we tested. Because it takes a long time to figure out how to design the grids and update it. Also, the amount of data processing will have a say for this problem. Like the amount of the US census data, we think sequential and O(n) time answering is good enough! If we were dealing with world population, parallelism and O(1) query will make much more sense!

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 talked about the process we take to find the four corners of US and its total population and we

choose to modify the rectangle class to give it a population field.

Then, we talked about the process updating the grid and float comparison.

Next, we talked about how to parallel combine the two grids.

Finally, we talked about lock and synchronous statements for version 5.

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

Yufang: V1, V2, debuging(Yeah!!)

Boren: V3, V4

Both: V5, writeup and experiment

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

It is fun to work together.

It’s extremely hard to debug someone’s code when you assume he initialized correctly.