1. *Who is in your group? (Only one writeup needs to be submitted per group.)*
   Nick Evans and Austin Briggs.

2. *What assistance did you receive on this project? Include anyone or anything except your partner, the course staff, and the course materials / textbook.*
   Wikipedia and searching StackOverflow.

3. *How long did the project take? Which parts were most difficult? How could the project be better?*
   The project took roughly half the time allotted (2 weeks). The most difficult parts were figuring out how to approach version 4 and debugging version 4. The project could be better if everything expected of us was clear enough on the spec that we would not have had to go to the message board.

4. *What (if any)"Above & Beyond" projects did you implement? What was interesting or difficult about them? Describe how you implemented them.*
   None.

5. *How did you test your program? What parts did you test in isolation and how? What smaller inputs did you create so that you could check your answers? What boundary cases did you consider?*
   For the most part we tested our program incrementally as we built it, using println statements to display the grid and verifying each part completed its task correctly before coding more.

   We also built our own mini census population txt file based off the example in version 3 of the spec to make the grid size more manageable, as well as the math when modifying the grid. This made it much easier to understand what values the elements of the grid were supposed to be as we built each version.

   We made sure to check that the error checking on the queries (e.g. east being smaller than west and whatnot) was correct by intentionally feeding the program invalid input. We also checked the boundary case of entering negative numbers as command line arguments and entering latitude values greater than 90 (this should not work).

   As we completed each version of our program, we checked to make sure it worked by inputting the sample values given on the discussion board.

MACHINE STATS FOR ?'s 6-8: Intel i7 – 3770 @ 3.40 GHz with 8 GB RAM on Windows 7

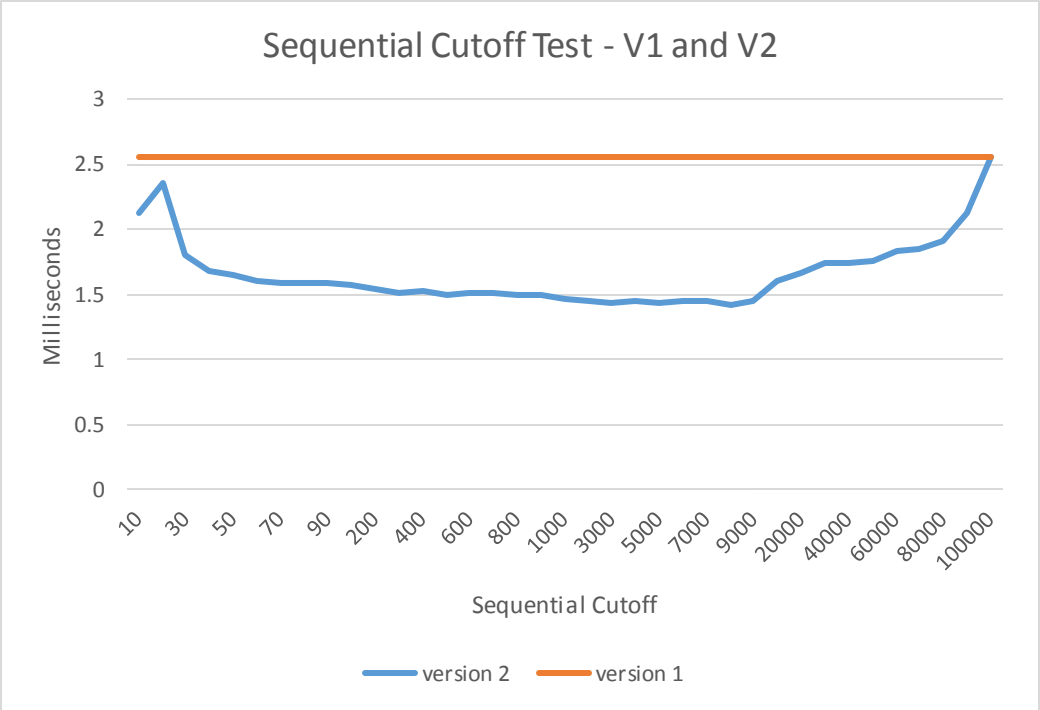
6. *For finding the corners of the United States and for the first grid-building step, you implemented parallel algorithms using Java's ForkJoin Framework. The code should have a sequential cut-off that can be varied. Perform experiments to determine the optimal value of this sequential cut-off. You need to examine cut-offs in two (really 3) places. First you are looking at sequential vs. parallel versions of corner finding.*

Looking at V1 and V2 works here, vary the cutoff for V2. Second you are looking at cut-offs in the grid-building step. Looking at V3 and V4 works here. There are two places to vary the cut-off for V4 - the the summing of census data and in the combining of grids. You don't have to worry about finding the optimal combination of cut-offs or anything but we do expect data on varying cut-offs for corner finding and both phases of grid building.

Graph the results and reach appropriate conclusions. 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?).

Part 1:
We predict that as we increase the sequential cutoff for the parallel square locator that the time of finding the corners of the United States will be exactly the same as sequentially finding the corners.

Conclusion: Our conclusion that as we increase the sequential cutoff was mostly correct. At the beginning since the sequential cutoff is small an increase in the cutoff resulted in a spike which can be seen at the twenty element cutoff. However after that point it dips down and decreases to the optimal point of 10000 elements in the sequential before it starts climbing toward the same number of milliseconds as the sequential algorithm

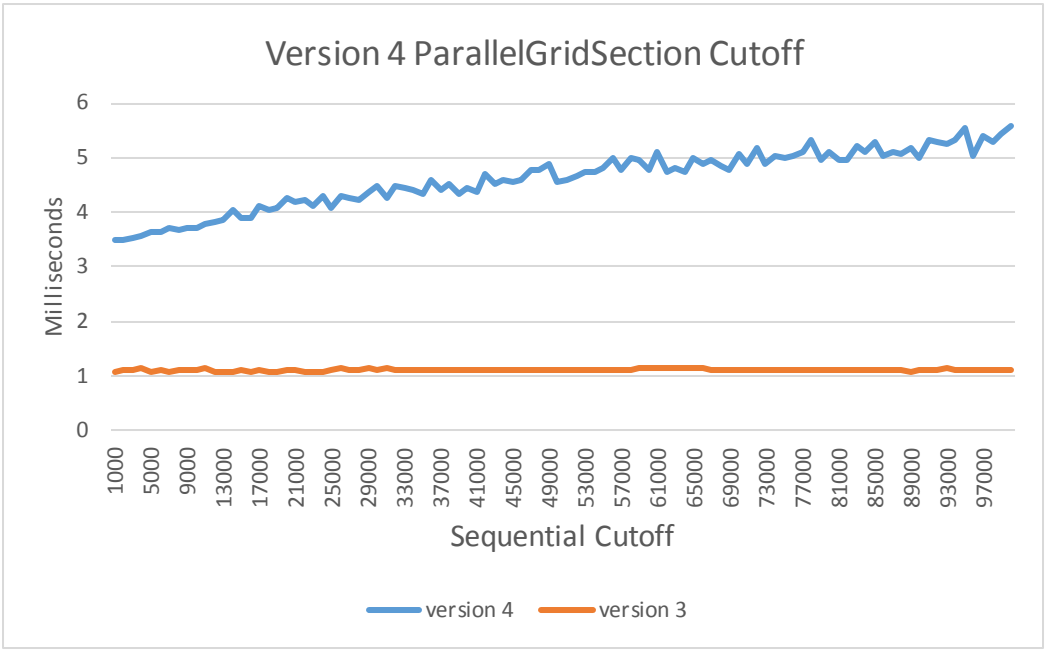


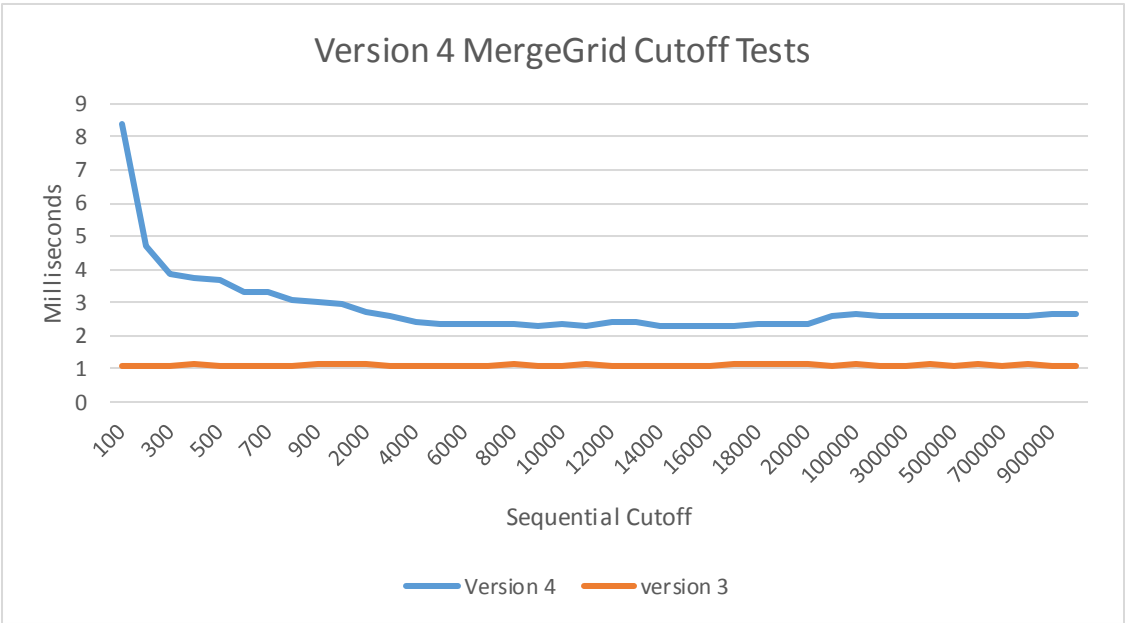
Part 2 graphs on the following page
Part 2A (Merge Grid's cutoff was set at 5000 for these tests):
We predict that the parallel creating will be more efficient than a sequential solution as we increase the sequential cutoff for the parallel grid section that the time of creating the grid of the United States will be exactly the same as sequentially.

Conclusion: Our prediction was not accurate. Version 3 was much more efficient in building than version 4 was. The likely reasons for our results are the cache not being used efficiently in Version 4, Version 4 taking up resource with creation of threads or creation of objects to determine location in version 4 rather than a direct calculation.

Part 2B (Parallel Grid Section cutoff set at 1000):
We predict that the parallel creating will be more efficient than a sequential solution as we increase the sequential cutoff for merge grid that the time of creating the grid of the United States will be exactly the same as sequentially. Also that for small values MergeGrid will not work and cause out of memory exceptions do to making too many threads during the MergeGrid portion.

Conclusion: As predicated with a small cutoff for Merge Grid we suffered out of memory exceptions. Also as we predicated with an increase in MergeGrid cutoff it became closer and closer to the sequential cutoff. However we were not accurate in predicating that modifying MergeGrid would be more efficient than sequentially. This can be attributed to adding multiple Grid Sections together and adding them sequentially which means adding together Grids that are equal in size to the Master Grid.

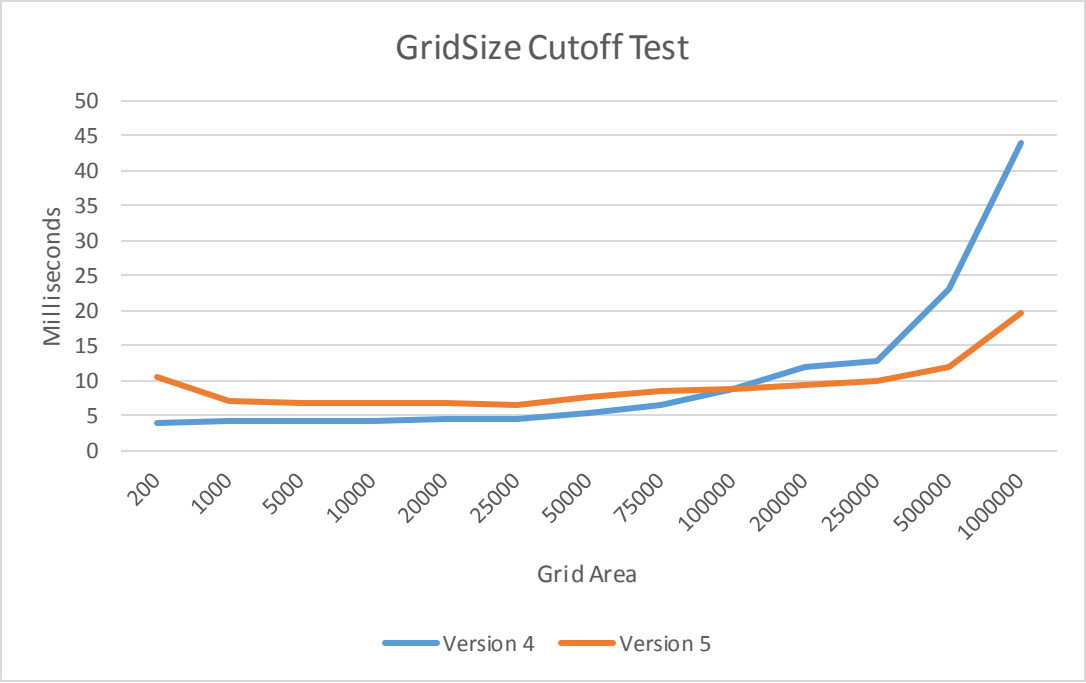

Version 4 ParallelGridSection Cutoff
Milliseconds
0
1
2
3
4
5
6
1000
5000
9000
13000
17000
21000
25000
29000
33000
37000
41000
45000
49000
53000
57000
61000
65000
69000
73000
77000
81000
85000
89000
93000
97000
Sequential Cutoff
version 4
version 3


Version 4 MergeGrid Cutoff Tests
Milliseconds
0
1
2
3
4
5
6
7
8
9
100
300
500
700
900
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
100000
300000
500000
700000
900000
Sequential Cutoff
Version 4
version 3

7. *Compare the performance of version 4 to version 5 as the size of the grid changes. Intuitively, which version is better for small grids and which version for large grids? Does the experimental data validate this hypothesis? Produce and interpret an appropriate graph or graphs to reach your conclusion.*

   Intuitively, version 4 should be faster than version 5 at smaller areas. This is because every time version 5 revisits a grid element, it may have to wait for the element to be unlocked before it can modify it. With smaller areas, there are less elements that will be modified more often so it makes sense that the total wait time for acquiring locks would be relatively higher. At higher areas, version 5 will have more elements that will modified less often, resulting in lower total wait time for acquiring locks. At this point, the grid-copying that version 4 does makes it slower than version 5.

   This intuition is verified by our data, as shown in the following graph. Version 5 starts beating version 4 at around an area of roughly 100,000.

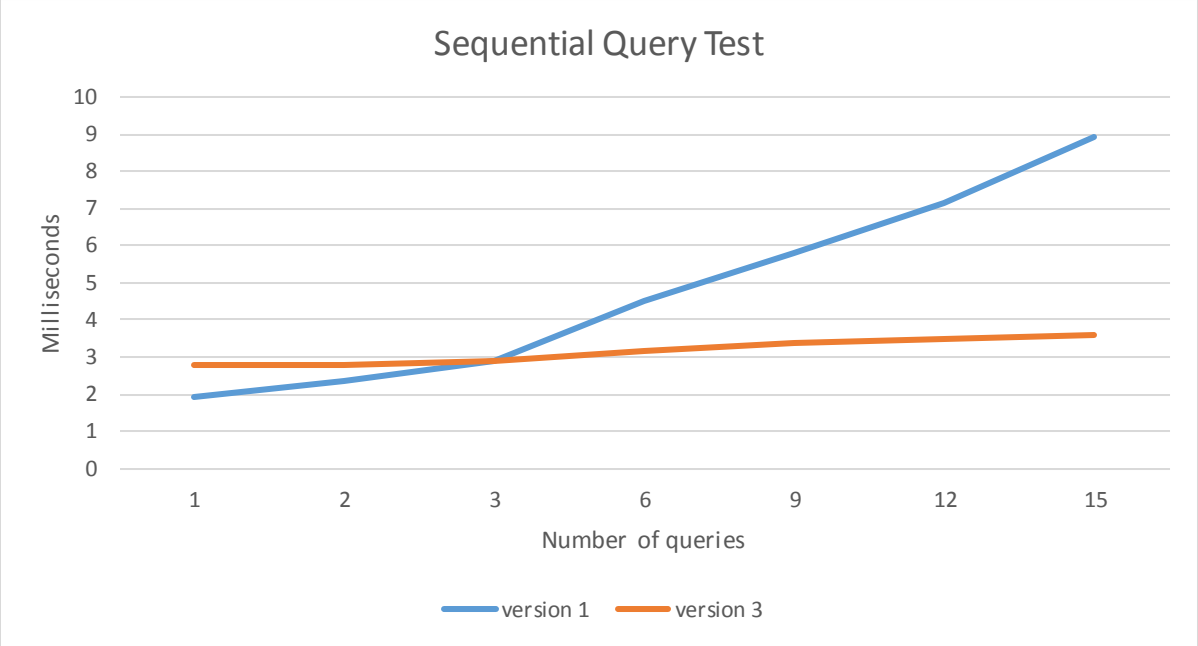


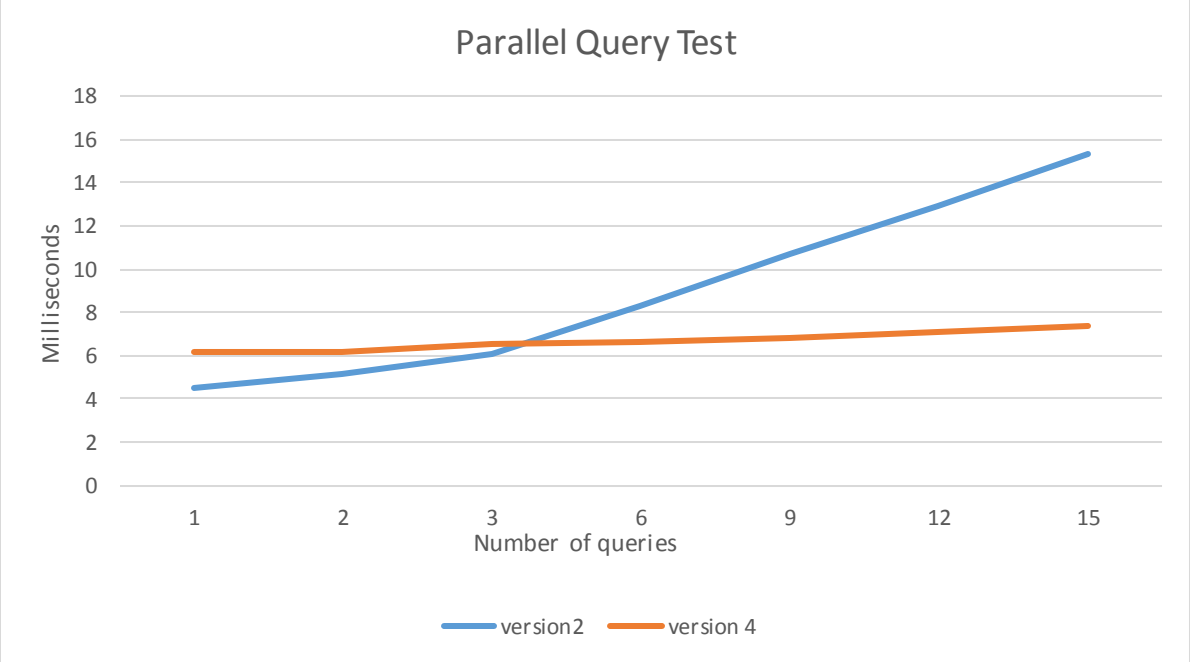
8. *Compare the performance of version 1 to version 3 and version 2 to version 4 as the number of queries changes. That is, how many queries are necessary before the pre-processing is worth it? Produce and interpret an appropriate graph or graphs to reach your conclusion. Note you should time the actual code answering the query, not including the time for a (very slow) human to enter the query.*

As shown in the "Sequential Query Test" graph, for the sequential tests on versions 1 and 3, version 3's extra pre-processing results in it beating version 1 starting at roughly 3 queries.

As shown in the "Parallel Query Test", for versions 2 and 4, version 4's extra pre-processing results in it beating version 2 starting at roughly 4 queries.

It is interesting to note that in both tests, the number of queries at the turning points was roughly the same.





9. *If you worked with a partner:*
    a. *Describe the process you used for developing and testing your code. If you divided it into pieces, describe that. If you worked on everything together, describe the actual process used. For example, discuss how long you talked about what, in what order you wrote and tested the code, and how long that required.*

        We assigned responsibilities by versions. Austin wrote versions 1 and 3, Nick wrote versions 2 and 4, and both of us worked on version 5. Both of us worked together for about 4-5 hours to figure out the approaches to versions 3 and 4. We both worked together to debug versions 3 and 4. Nick wrote the testing code for the writeup and we both ran the simulations together to gather data. Both of us worked together to answer the writeup questions.

        On the whole, we worked on the versions in order since later versions were able to use code from prior versions. We wrote and verified correctness of each version before moving on to the next version. The entire process took about 2 weeks' worth of work.

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

        See first paragraph of part a.'s response.

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

        One bad thing was that Nick had to wait on Austin for him to finish up his versions so Nick could reuse the applicable code from the versions Austin worked on. One good thing was that working together allowed us to bounce ideas off each other to figure out tougher sections like version 4. It also helped break down the workload to work with a partner.