

Peri Hassanzadeh
ECE1570 High Performance Computing
Dr. Bigrigg
November 5, 2021

Threads and Parallel Programming

Introduction

In this assignment, I was able to use threads and develop data and task parallel algorithms. Utilizing pthreads and ideas such as busy-wait and mutexes, I was able to develop a better understanding of programming in parallel and how this could be used in other applications as well. In addition, I identified the speedup I could achieve by using multiple processors on the bigrigg.com server.

The main objective of each program no matter it's algorithm was to read and perform actions on certain word documents. I was to read in the data from the text files, split the characters and filter out noise words as well as whitespace, symbols, and numbers. After performing certain actions on the read in text, it was sent to another document in which the frequency of each word was found and then in a final report, the top 10% of most frequent words were listed in alphabetical order.

In order to run each file, make sure that all .cpp files are in the same folder as the working directory. A folder named "certdata" should also be in the working directory with all of the text files within it. This format will allow for successful compilation of each implementation resulting in an executable file with the following commands:

Serial command:

```
g++ serial.cpp -o serial
```

Data Parallel command:

```
Windows: g++ dataParallel.cpp -o dataParallel -lpthread
```

```
Linux: g++ dataParallel.cpp -o dataParallel -pthread
```

Task Parallel command:

```
Windows: g++ taskParallel.cpp -o taskParallel -lpthread
```

```
Linux: g++ taskParallel.cpp -o taskParallel -pthread
```

Finally, note that each implementation could be optimized in various ways. I will go over a few points that I wish I would have been able to develop on given better time management.

Serial Version

The files that correspond with the serial version are as follows:

serial.cpp - implementation of serial version

serial.exe - compiled program file

results.txt - file with all of the filtered words listed from every text file

finalreport.txt - summary file with top 10% of most used words listed

Screenshots of Code:

```
1 //Serial Implementation
2 //ECE157N High Performance Computing
3 //Date of Creation: 20/10/2021
4 //Last Update: 11/5/2021
5
6 //This is the serial version of Project 1, opening text documents,
7 //filtering the words, counting the frequency, and summarizing the
8 //top 10% of most frequent concepts.
9
10 //Import C++ Libraries
11 #include <iostream>
12 #include <fstream>
13 #include <bits/stdc++.h>
14 #include <string>
15 #include <sys/time.h>
16 using namespace std;
17
18
19 /**
20 * Prints the frequency of the top 10% most frequently used words and sends them to a final report.
21 * This method was based off of an online resource that will be listed at the end of this document. (1)
22 */
23 void printFreq()
24 {
25     //Initialize variables
26
27     //wordVec to hold the words from the results.txt file
28     vector<string> wordVec;
29
30     //resultfile to read in the filtered words from a file
31     ifstream resultfile;
32     //finalreport to store the resulting frequency calculation to the final summary text file
33     ofstream finalreport;
34
35     //Open the subsequent files for input and output
36     resultfile.open("results.txt");
37     finalreport.open("finalreport.txt");
38
39     //String to hold the words from results.txt file
40     string words;
41
42     //Until the end of the file, read in each line and store it in words variable
43     while(getline(resultfile, words))
44     {
45         //Add the words from each line into a vector
46         wordVec.push_back(words);
47     }
48
49     //Create a map of a string and an integer value
50     //The string holds the word and the integer is the frequency
51     map<string, int> M;
52
53     //Iterate through each word in the vector of all words
54     for(int i=0; i<wordVec.size(); i++)
55     {
56         if(M.find(wordVec[i]) == M.end())
57         {
58             //If the word is only found once it is end of map then the count is one
59             M[wordVec[i]] = 1;
60         }
61         else
62         {
63             //Increment the count of the frequency in the map
64             M[wordVec[i]]++;
65         }
66     }
67 }
```

```
112 //Iterate through each file
113 for(int i=0; i<files.size(); i++)
114 {
115     //Open the corresponding text file
116     infile.open(files[i]);
117
118     //Read each line of the text file until the end of the file and store it in the string words
119     while (getline(infile, words)) {
120         //Remove symbols and numbers from the line
121         for (int j=0; j<words.size(); j++)
122             if (!isalpha(words[j]))
123                 words[j] = ' ';
124     }
125
126     //Make sure each letter is lowercase in the string words
127     for (int i=0; i<words.size(); i++)
128         words[i] = tolower(words[i]);
129
130     //Make into a list of words
131     vector<string> wordlist;
132     char* charwords = new char[words.length()+1];
133     strcpy(charwords, words.c_str());
134     //Iterate through the string of words to know a new word is found
135     char* token = strtok(charwords, " ");
136     while (token != NULL)
137     {
138         //Add the token to the list of words
139         wordlist.push_back(token);
140         token = strtok(NULL, " ");
141     }
142
143     //Find and remove the noise words
144     for (int k=wordlist.size()-1; k>=0; k--)
145     {
146         //Indicator to remove
147         int remove = 0;
148         //Iterate the wordlist vector looking for each noise word
149         for (int n=0; n<delwords.size(); n++)
150         {
151             if (wordlist[k].compare(delwords[n]) == 0)
152             {
153                 remove = 1;
154             }
155         }
156         if (remove == 1) // If the
157             wordlist.erase(wordlist.begin()+k);
158     }
159
160     //For each word in the wordlist vector, print it to a new line in the results.txt file
161     for (int n=0; n<wordlist.size(); n++)
162         outfile << wordlist[n] << endl;
163 }
164
165 //Close the file that the words were being read from in order to open the next file
166 infile.close();
167
168 //Close the file that the words were being printed to "results.txt"
169 outfile.close();
170
171 //Call the method to print the final summary and frequency of words
172 printFreq();
173
174 //Get the time the program finishes at
175 gettimeofday(&end, NULL);
176
177 //Calculate the time that was taken for the program to complete and print out the results (3)
178 int micro_end = end.tv_sec * 1000000 + end.tv_usec;
179 int micro_start = start.tv_sec * 1000000 + start.tv_usec;
180 cout << "Total time for serial execution: " << (micro_end - micro_start) << " microseconds" << endl;
181 return 0;
182 }
```

```
67 //size is the total amount of words found
68 int size = wordVec.size();
69 //size is the percent of all the words found
70 int temp = ceil(size*.1);
71
72 //Create an iterator to go through each word in the map
73 for(auto it=M.begin(); it!=M.end(); it++)
74 {
75     //If the frequency of the word is more than 10 send it to the final report summary
76     if(it.second > 10)
77         finalreport << it.first << " " << it.second << '\n';
78 }
79
80 /**
81 * Main function for the majority of the text analysis.
82 * This method was based off of an online resource that will be listed at the end of this document. (2)
83 */
84 int main()
85 {
86     //Structure to hold the variables in order to find completion time
87     struct timeval start, end;
88     //Get the time the program starts
89     gettimeofday(&start, NULL);
90
91     //Iterate through each data from the given text files in certdata
92     ifstream infile;
93     //outfile to print out the results of the filtered words to results.txt
94     ofstream outfile;
95     outfile.open("results.txt");
96
97     //wordVec to hold all of the filtered words to go into the results.txt file
98     vector<string> wordVec;
99
100     //String to hold each line of the text file
101     string words;
102
103     //Array to hold the names of all of the files
104     const char* files[70] = {
105         "certdata/2013-01-10.txt", "certdata/2013-01-09.txt", "certdata/2013-01-08.txt", "certdata/2013-01-07.txt", "certdata/2013-01-06.txt", "certdata/2013-01-05.txt", "certdata/2013-01-04.txt", "certdata/2013-01-03.txt", "certdata/2013-01-02.txt", "certdata/2013-01-01.txt", "certdata/2013-12-31.txt", "certdata/2013-12-30.txt", "certdata/2013-12-29.txt", "certdata/2013-12-28.txt", "certdata/2013-12-27.txt", "certdata/2013-12-26.txt", "certdata/2013-12-25.txt", "certdata/2013-12-24.txt", "certdata/2013-12-23.txt", "certdata/2013-12-22.txt", "certdata/2013-12-21.txt", "certdata/2013-12-20.txt", "certdata/2013-12-19.txt", "certdata/2013-12-18.txt", "certdata/2013-12-17.txt", "certdata/2013-12-16.txt", "certdata/2013-12-15.txt", "certdata/2013-12-14.txt", "certdata/2013-12-13.txt", "certdata/2013-12-12.txt", "certdata/2013-12-11.txt", "certdata/2013-12-10.txt", "certdata/2013-12-09.txt", "certdata/2013-12-08.txt", "certdata/2013-12-07.txt", "certdata/2013-12-06.txt", "certdata/2013-12-05.txt", "certdata/2013-12-04.txt", "certdata/2013-12-03.txt", "certdata/2013-12-02.txt", "certdata/2013-12-01.txt", "certdata/2013-11-30.txt", "certdata/2013-11-29.txt", "certdata/2013-11-28.txt", "certdata/2013-11-27.txt", "certdata/2013-11-26.txt", "certdata/2013-11-25.txt", "certdata/2013-11-24.txt", "certdata/2013-11-23.txt", "certdata/2013-11-22.txt", "certdata/2013-11-21.txt", "certdata/2013-11-20.txt", "certdata/2013-11-19.txt", "certdata/2013-11-18.txt", "certdata/2013-11-17.txt", "certdata/2013-11-16.txt", "certdata/2013-11-15.txt", "certdata/2013-11-14.txt", "certdata/2013-11-13.txt", "certdata/2013-11-12.txt", "certdata/2013-11-11.txt", "certdata/2013-11-10.txt", "certdata/2013-11-09.txt", "certdata/2013-11-08.txt", "certdata/2013-11-07.txt", "certdata/2013-11-06.txt", "certdata/2013-11-05.txt", "certdata/2013-11-04.txt", "certdata/2013-11-03.txt", "certdata/2013-11-02.txt", "certdata/2013-11-01.txt", "certdata/2013-10-31.txt", "certdata/2013-10-30.txt", "certdata/2013-10-29.txt", "certdata/2013-10-28.txt", "certdata/2013-10-27.txt", "certdata/2013-10-26.txt", "certdata/2013-10-25.txt", "certdata/2013-10-24.txt", "certdata/2013-10-23.txt", "certdata/2013-10-22.txt", "certdata/2013-10-21.txt", "certdata/2013-10-20.txt", "certdata/2013-10-19.txt", "certdata/2013-10-18.txt", "certdata/2013-10-17.txt", "certdata/2013-10-16.txt", "certdata/2013-10-15.txt", "certdata/2013-10-14.txt", "certdata/2013-10-13.txt", "certdata/2013-10-12.txt", "certdata/2013-10-11.txt", "certdata/2013-10-10.txt", "certdata/2013-10-09.txt", "certdata/2013-10-08.txt", "certdata/2013-10-07.txt", "certdata/2013-10-06.txt", "certdata/2013-10-05.txt", "certdata/2013-10-04.txt", "certdata/2013-10-03.txt", "certdata/2013-10-02.txt", "certdata/2013-10-01.txt", "certdata/2013-09-30.txt", "certdata/2013-09-29.txt", "certdata/2013-09-28.txt", "certdata/2013-09-27.txt", "certdata/2013-09-26.txt", "certdata/2013-09-25.txt", "certdata/2013-09-24.txt", "certdata/2013-09-23.txt", "certdata/2013-09-22.txt", "certdata/2013-09-21.txt", "certdata/2013-09-20.txt", "certdata/2013-09-19.txt", "certdata/2013-09-18.txt", "certdata/2013-09-17.txt", "certdata/2013-09-16.txt", "certdata/2013-09-15.txt", "certdata/2013-09-14.txt", "certdata/2013-09-13.txt", "certdata/2013-09-12.txt", "certdata/2013-09-11.txt", "certdata/2013-09-10.txt", "certdata/2013-09-09.txt", "certdata/2013-09-08.txt", "certdata/2013-09-07.txt", "certdata/2013-09-06.txt", "certdata/2013-09-05.txt", "certdata/2013-09-04.txt", "certdata/2013-09-03.txt", "certdata/2013-09-02.txt", "certdata/2013-09-01.txt", "certdata/2013-08-31.txt", "certdata/2013-08-30.txt", "certdata/2013-08-29.txt", "certdata/2013-08-28.txt", "certdata/2013-08-27.txt", "certdata/2013-08-26.txt", "certdata/2013-08-25.txt", "certdata/2013-08-24.txt", "certdata/2013-08-23.txt", "certdata/2013-08-22.txt", "certdata/2013-08-21.txt", "certdata/2013-08-20.txt", "certdata/2013-08-19.txt", "certdata/2013-08-18.txt", "certdata/2013-08-17.txt", "certdata/2013-08-16.txt", "certdata/2013-08-15.txt", "certdata/2013-08-14.txt", "certdata/2013-08-13.txt", "certdata/2013-08-12.txt", "certdata/2013-08-11.txt", "certdata/2013-08-10.txt", "certdata/2013-08-09.txt", "certdata/2013-08-08.txt", "certdata/2013-08-07.txt", "certdata/2013-08-06.txt", "certdata/2013-08-05.txt", "certdata/2013-08-04.txt", "certdata/2013-08-03.txt", "certdata/2013-08-02.txt", "certdata/2013-08-01.txt", "certdata/2013-07-31.txt", "certdata/2013-07-30.txt", "certdata/2013-07-29.txt", "certdata/2013-07-28.txt", "certdata/2013-07-27.txt", "certdata/2013-07-26.txt", "certdata/2013-07-25.txt", "certdata/2013-07-24.txt", "certdata/2013-07-23.txt", "certdata/2013-07-22.txt", "certdata/2013-07-21.txt", "certdata/2013-07-20.txt", "certdata/2013-07-19.txt", "certdata/2013-07-18.txt", "certdata/2013-07-17.txt", "certdata/2013-07-16.txt", "certdata/2013-07-15.txt", "certdata/2013-07-14.txt", "certdata/2013-07-13.txt", "certdata/2013-07-12.txt", "certdata/2013-07-11.txt", "certdata/2013-07-10.txt", "certdata/2013-07-09.txt", "certdata/2013-07-08.txt", "certdata/2013-07-07.txt", "certdata/2013-07-06.txt", "certdata/2013-07-05.txt", "certdata/2013-07-04.txt", "certdata/2013-07-03.txt", "certdata/2013-07-02.txt", "certdata/2013-07-01.txt", "certdata/2013-06-30.txt", "certdata/2013-06-29.txt", "certdata/2013-06-28.txt", "certdata/2013-06-27.txt", "certdata/2013-06-26.txt", "certdata/2013-06-25.txt", "certdata/2013-06-24.txt", "certdata/2013-06-23.txt", "certdata/2013-06-22.txt", "certdata/2013-06-21.txt", "certdata/2013-06-20.txt", "certdata/2013-06-19.txt", "certdata/2013-06-18.txt", "certdata/2013-06-17.txt", "certdata/2013-06-16.txt", "certdata/2013-06-15.txt", "certdata/2013-06-14.txt", "certdata/2013-06-13.txt", "certdata/2013-06-12.txt", "certdata/2013-06-11.txt", "certdata/2013-06-10.txt", "certdata/2013-06-09.txt", "certdata/2013-06-08.txt", "certdata/2013-06-07.txt", "certdata/2013-06-06.txt", "certdata/2013-06-05.txt", "certdata/2013-06-04.txt", "certdata/2013-06-03.txt", "certdata/2013-06-02.txt", "certdata/2013-06-01.txt", "certdata/2013-05-31.txt", "certdata/2013-05-30.txt", "certdata/2013-05-29.txt", "certdata/2013-05-28.txt", "certdata/2013-05-27.txt", "certdata/2013-05-26.txt", "certdata/2013-05-25.txt", "certdata/2013-05-24.txt", "certdata/2013-05-23.txt", "certdata/2013-05-22.txt", "certdata/2013-05-21.txt", "certdata/2013-05-20.txt", "certdata/2013-05-19.txt", "certdata/2013-05-18.txt", "certdata/2013-05-17.txt", "certdata/2013-05-16.txt", "certdata/2013-05-15.txt", "certdata/2013-05-14.txt", "certdata/2013-05-13.txt", "certdata/2013-05-12.txt", "certdata/2013-05-11.txt", "certdata/2013-05-10.txt", "certdata/2013-05-09.txt", "certdata/2013-05-08.txt", "certdata/2013-05-07.txt", "certdata/2013-05-06.txt", "certdata/2013-05-05.txt", "certdata/2013-05-04.txt", "certdata/2013-05-03.txt", "certdata/2013-05-02.txt", "certdata/2013-05-01.txt", "certdata/2013-04-30.txt", "certdata/2013-04-29.txt", "certdata/2013-04-28.txt", "certdata/2013-04-27.txt", "certdata/2013-04-26.txt", "certdata/2013-04-25.txt", "certdata/2013-04-24.txt", "certdata/2013-04-23.txt", "certdata/2013-04-22.txt", "certdata/2013-04-21.txt", "certdata/2013-04-20.txt", "certdata/2013-04-19.txt", "certdata/2013-04-18.txt", "certdata/2013-04-17.txt", "certdata/2013-04-16.txt", "certdata/2013-04-15.txt", "certdata/2013-04-14.txt", "certdata/2013-04-13.txt", "certdata/2013-04-12.txt", "certdata/2013-04-11.txt", "certdata/2013-04-10.txt", "certdata/2013-04-09.txt", "certdata/2013-04-08.txt", "certdata/2013-04-07.txt", "certdata/2013-04-06.txt", "certdata/2013-04-05.txt", "certdata/2013-04-04.txt", "certdata/2013-04-03.txt", "certdata/2013-04-02.txt", "certdata/2013-04-01.txt", "certdata/2013-03-31.txt", "certdata/2013-03-30.txt", "certdata/2013-03-29.txt", "certdata/2013-03-28.txt", "certdata/2013-03-27.txt", "certdata/2013-03-26.txt", "certdata/2013-03-25.txt", "certdata/2013-03-24.txt", "certdata/2013-03-23.txt", "certdata/2013-03-22.txt", "certdata/2013-03-21.txt", "certdata/2013-03-20.txt", "certdata/2013-03-19.txt", "certdata/2013-03-18.txt", "certdata/2013-03-17.txt", "certdata/2013-03-16.txt", "certdata/2013-03-15.txt", "certdata/2013-03-14.txt", "certdata/2013-03-13.txt", "certdata/2013-03-12.txt", "certdata/2013-03-11.txt", "certdata/2013-03-10.txt", "certdata/2013-03-09.txt", "certdata/2013-03-08.txt", "certdata/2013-03-07.txt", "certdata/2013-03-06.txt", "certdata/2013-03-05.txt", "certdata/2013-03-04.txt", "certdata/2013-03-03.txt", "certdata/2013-03-02.txt", "certdata/2013-03-01.txt", "certdata/2013-02-28.txt", "certdata/2013-02-27.txt", "certdata/2013-02-26.txt", "certdata/2013-02-25.txt", "certdata/2013-02-24.txt", "certdata/2013-02-23.txt", "certdata/2013-02-22.txt", "certdata/2013-02-21.txt", "certdata/2013-02-20.txt", "certdata/2013-02-19.txt", "certdata/2013-02-18.txt", "certdata/2013-02-17.txt", "certdata/2013-02-16.txt", "certdata/2013-02-15.txt", "certdata/2013-02-14.txt", "certdata/2013-02-13.txt", "certdata/2013-02-12.txt", "certdata/2013-02-11.txt", "certdata/2013-02-10.txt", "certdata/2013-02-09.txt", "certdata/2013-02-08.txt", "certdata/2013-02-07.txt", "certdata/2013-02-06.txt", "certdata/2013-02-05.txt", "certdata/2013-02-04.txt", "certdata/2013-02-03.txt", "certdata/2013-02-02.txt", "certdata/2013-02-01.txt", "certdata/2013-01-31.txt", "certdata/2013-01-30.txt", "certdata/2013-01-29.txt", "certdata/2013-01-28.txt", "certdata/2013-01-27.txt", "certdata/2013-01-26.txt", "certdata/2013-01-25.txt", "certdata/2013-01-24.txt", "certdata/2013-01-23.txt", "certdata/2013-01-22.txt", "certdata/2013-01-21.txt", "certdata/2013-01-20.txt", "certdata/2013-01-19.txt", "certdata/2013-01-18.txt", "certdata/2013-01-17.txt", "certdata/2013-01-16.txt", "certdata/2013-01-15.txt", "certdata/2013-01-14.txt", "certdata/2013-01-13.txt", "certdata/2013-01-12.txt", "certdata/2013-01-11.txt", "certdata/2013-01-10.txt", "certdata/2013-01-09.txt", "certdata/2013-01-08.txt", "certdata/2013-01-07.txt", "certdata/2013-01-06.txt", "certdata/2013-01-05.txt", "certdata/2013-01-04.txt", "certdata/2013-01-03.txt", "certdata/2013-01-02.txt", "certdata/2013-01-01.txt", "certdata/2012-12-31.txt", "certdata/2012-12-30.txt", "certdata/2012-12-29.txt", "certdata/2012-12-28.txt", "certdata/2012-12-27.txt", "certdata/2012-12-26.txt", "certdata/2012-12-25.txt", "certdata/2012-12-24.txt", "certdata/2012-12-23.txt", "certdata/2012-12-22.txt", "certdata/2012-12-21.txt", "certdata/2012-12-20.txt", "certdata/2012-12-19.txt", "certdata/2012-12-18.txt", "certdata/2012-12-17.txt", "certdata/2012-12-16.txt", "certdata/2012-12-15.txt", "certdata/2012-12-14.txt", "certdata/2012-12-13.txt", "certdata/2012-12-12.txt", "certdata/2012-12-11.txt", "certdata/2012-12-10.txt", "certdata/2012-12-09.txt", "certdata/2012-12-08.txt", "certdata/2012-12-07.txt", "certdata/2012-12-06.txt", "certdata/2012-12-05.txt", "certdata/2012-12-04.txt", "certdata/2012-12-03.txt", "certdata/2012-12-02.txt", "certdata/2012-12-01.txt", "certdata/2012-11-30.txt", "certdata/2012-11-29.txt", "certdata/2012-11-28.txt", "certdata/2012-11-27.txt", "certdata/2012-11-26.txt", "certdata/2012-11-25.txt", "certdata/2012-11-24.txt", "certdata/2012-11-23.txt", "certdata/2012-11-22.txt", "certdata/2012-11-21.txt", "certdata/2012-11-20.txt", "certdata/2012-11-19.txt", "certdata/2012-11-18.txt", "certdata/2012-11-17.txt", "certdata/2012-11-16.txt", "certdata/2012-11-15.txt", "certdata/2012-11-14.txt", "certdata/2012-11-13.txt", "certdata/2012-11-12.txt", "certdata/2012-11-11.txt", "certdata/2012-11-10.txt", "certdata/2012-11-09.txt", "certdata/2012-11-08.txt", "certdata/2012-11-07.txt", "certdata/2012-11-06.txt", "certdata/2012-11-05.txt", "certdata/2012-11-04.txt", "certdata/2012-11-03.txt", "certdata/2012-11-02.txt", "certdata/2012-11-01.txt", "certdata/2012-10-31.txt", "certdata/2012-10-30.txt", "certdata/2012-10-29.txt", "certdata/2012-10-28.txt", "certdata/2012-10-27.txt", "certdata/2012-10-26.txt", "certdata/2012-10-25.txt", "certdata/2012-10-24.txt", "certdata/2012-10-23.txt", "certdata/2012-10-22.txt", "certdata/2012-10-21.txt", "certdata/2012-10-20.txt", "certdata/2012-10-19.txt", "certdata/2012-10-18.txt", "certdata/2012-10-17.txt", "certdata/2012-10-16.txt", "certdata/2012-10-15.txt", "certdata/2012-10-14.txt", "certdata/2012-10-13.txt", "certdata/2012-10-12.txt", "certdata/2012-10-11.txt", "certdata/2012-10-10.txt", "certdata/2012-10-09.txt", "certdata/2012-10-08.txt", "certdata/2012-10-07.txt", "certdata/2012-10-06.txt", "certdata/2012-10-05.txt", "certdata/2012-10-04.txt", "certdata/2012-10-03.txt", "certdata/2012-10-02.txt", "certdata/2012-10-01.txt", "certdata/2012-09-30.txt", "certdata/2012-09-29.txt", "certdata/2012-09-28.txt", "certdata/2012-09-27.txt", "certdata/2012-09-26.txt", "certdata/2012-09-25.txt", "certdata/2012-09-24.txt", "certdata/2012-09-23.txt", "certdata/2012-09-22.txt", "certdata/2012-09-21.txt", "certdata/2012-09-20.txt", "certdata/2012-09-19.txt", "certdata/2012-09-18.txt", "certdata/2012-09-17.txt", "certdata/2012-09-16.txt", "certdata/2012-09-15.txt", "certdata/2012-09-14.txt", "certdata/2012-09-13.txt", "certdata/2012-09-12.txt", "certdata/2012-09-11.txt", "certdata/2012-09-10.txt", "certdata/2012-09-09.txt", "certdata/2012-09-08.txt", "certdata/2012-09-07.txt", "certdata/2012-09-06.txt", "certdata/2012-09-05.txt", "certdata/2012-09-04.txt", "certdata/2012-09-03.txt", "certdata/2012-09-02.txt", "certdata/2012-09-01.txt", "certdata/2012-08-31.txt", "certdata/2012-08-30.txt", "certdata/2012-08-29.txt", "certdata/2012-08-28.txt", "certdata/2012-08-27.txt", "certdata/2012-08-26.txt", "certdata/2012-08-25.txt", "certdata/2012-08-24.txt", "certdata/2012-08-23.txt", "certdata/2012-08-22.txt", "certdata/2012-08-21.txt", "certdata/2012-08-20.txt", "certdata/2012-08-19.txt", "certdata/2012-08-18.txt", "certdata/2012-08-17.txt", "certdata/2012-08-16.txt", "certdata/2012-08-15.txt", "certdata/2012-08-14.txt", "certdata/2012-08-13.txt", "certdata/2012-08-12.txt", "certdata/2012-08-11.txt", "certdata/2012-08-10.txt", "certdata/2012-08-09.txt", "certdata/2012-08-08.txt", "certdata/2012-08-07.txt", "certdata/2012-08-06.txt", "certdata/2012-08-05.txt", "certdata/2012-08-04.txt", "certdata/2012-08-03.txt", "certdata/2012-08-02.txt", "certdata/2012-08-01.txt", "certdata/2012-07-31.txt", "certdata/2012-07-30.txt", "certdata/2012-07-29.txt", "certdata/2012-07-28.txt", "certdata/2012-07-27.txt", "certdata/2012-07-26.txt", "certdata/2012-07-25.txt", "certdata/2012-07-24.txt", "certdata/2012-07-23.txt", "certdata/2012-07-22.txt", "certdata/2012-07-21.txt", "certdata/2012-07-20.txt", "certdata/2012-07-19.txt", "certdata/2012-07-18.txt", "certdata/2012-07-17.txt", "certdata/2012-07-16.txt", "certdata/2012-07-15.txt", "certdata/2012-07-14.txt", "certdata/2012-07-13.txt", "certdata/2012-07-12.txt", "certdata/2012-07-11.txt", "certdata/2012-07-10.txt", "certdata/2012-07-09.txt", "certdata/2012-07-08.txt", "certdata/2012-07-07.txt", "certdata/2012-07-06.txt", "certdata/2012-07-05.txt", "certdata/2012-07-04.txt", "certdata/2012-07-03.txt", "certdata/2012-07-02.txt", "certdata/2012-07-01.txt", "certdata/2012-06-30.txt", "certdata/2012-06-29.txt", "certdata/2012-06-28.txt", "certdata/2012-06-27.txt", "certdata/2012-06-26.txt", "certdata/2012-06-25.txt", "certdata/2012-06-24.txt", "certdata/2012-06-23.txt", "certdata/2012-06-22.txt", "certdata/2012-06-21.txt", "certdata/2012-06-20.txt", "certdata/2012-06-19.txt", "certdata/2012-06-18.txt", "certdata/2012-06-17.txt", "certdata/2012-06-16.txt", "certdata/2012-06-15.txt", "certdata/2012-06-14.txt", "certdata/2012-06-13.txt", "certdata/2012-06-12.txt", "certdata/2012-06-11.txt", "certdata/2012-06-10.txt", "certdata/2012-06-09.txt", "certdata/2012-06-08.txt", "certdata/2012-06-07.txt", "certdata/2012-06-06.txt", "certdata/2012-06-05.txt", "certdata/2012-06-04.txt", "certdata/2012-06-03.txt", "certdata/2012-06-02.txt", "certdata/2012-06-01.txt", "certdata/2012-05-31.txt", "certdata/2012-05-30.txt", "certdata/2012-05-29.txt", "certdata/2012-05-28.txt", "certdata/2012-05-27.txt", "certdata/2012-05-26.txt", "certdata/2012-05-25.txt", "certdata/2012-05-24.txt", "certdata/2012-05-23.txt", "certdata/2012-05-22.txt", "certdata/2012-05-21.txt", "certdata/2012-05-20.txt", "certdata/2012-05-19.txt", "certdata/2012-05-18.txt", "certdata/2012-05-17.txt", "certdata/2012-05-16.txt", "certdata/2012-05-15.txt", "certdata/2012-05-14.txt", "certdata/2012-05-13.txt", "certdata/2012-05-12.txt", "certdata/2012-05-11.txt", "certdata/2012-05-10.txt", "certdata/2012-05-09.txt", "certdata/2012-05-08.txt", "certdata/2012-05-07.txt", "certdata/2012-05-06.txt", "certdata/2012-05-05.txt", "certdata/2012-05-04.txt", "certdata/2012-05-03.txt", "certdata/2012-05-02.txt", "certdata/2012-05-01.txt", "certdata/2012-04-30.txt", "certdata/2012-04-29.txt", "certdata/2012-04-28.txt", "certdata/2012-04-27.txt",
```

Output:

From my local machine:

```
C:\Users\perih\Downloads\ECE1570_Project1\ECE1570_Project1>serial
Total time for serial execution: 269864 microseconds
```

finalreport.txt - Notepad

File Edit Format View Help

unauthenticated 46

unauthorized 17

update 14

url 16

use 24

used 48

user 82

users 11

using 28

value 13

version 45

versions 40

via 19

view 12

vulnerabilities 56

vulnerability 126

vulnerable 44

web 69

when 19

which 46

will 27

windows 19

wireless 14

within 20

without 14

www 12

x 148

xml 26

xss 11

zip 31

<

Ln 209, Col 7

The execution of the program took 269864 microseconds on my local machine. In the text file, it specifies there are 209 lines meaning that the top 10% of most used words summary file listed 209 different words. The results should be the same for the rest of the files if they were successful.

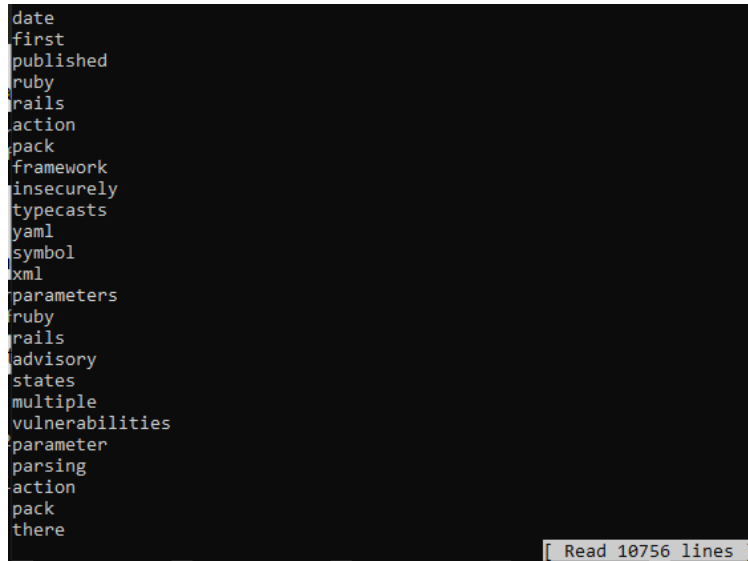
From Bigrigg Server:

Execution time

```
[northampton]$ ./serial
Total time for serial execution: 123376 microseconds
[northampton]$
```

From the ssh server bigrigg.com, the execution took 123376 microseconds which is significantly less than that of my local machine. Additionally, I will make sure to analyze the rest of the execution times for data parallel and task parallel from the server as well as my local machine. Comparing the two values will allow me to properly decide proper scaling and different implementation benefits and drawbacks.

results.txt

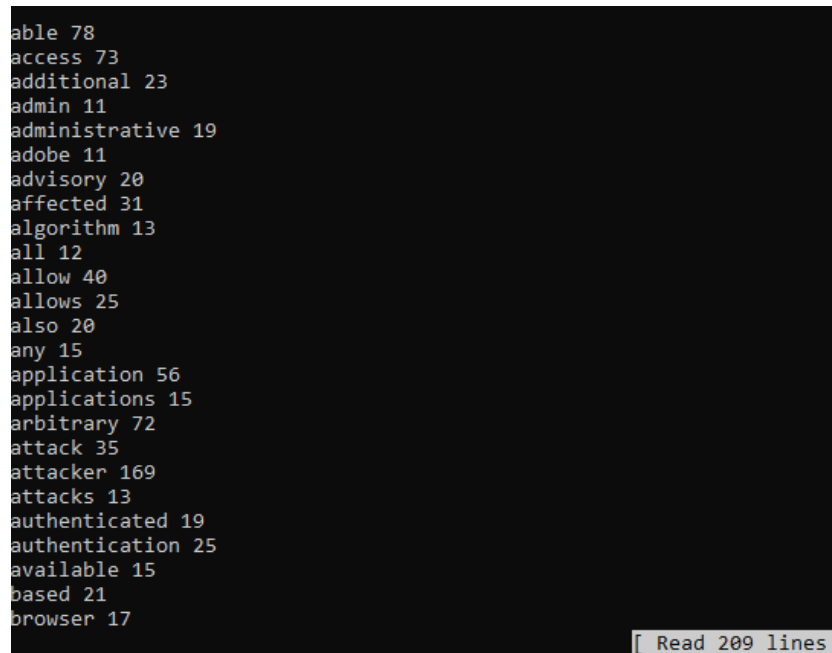


```
date
first
published
ruby
rails
action
pack
framework
insecurely
typecasts
yaml
symbol
xml
parameters
ruby
rails
advisory
states
multiple
vulnerabilities
parameter
parsing
action
pack
there
```

[Read 10756 lines]

The results.txt file holds all of the filtered words before the frequency is counted and analyzed by the top 10% of most frequently used words. Currently there are 10756 total words as listed by the number of lines since there is one word per line.

finalresults.txt



```
able 78
access 73
additional 23
admin 11
administrative 19
adobe 11
advisory 20
affected 31
algorithm 13
all 12
allow 40
allows 25
also 20
any 15
application 56
applications 15
arbitrary 72
attack 35
attacker 169
attacks 13
authenticated 19
authentication 25
available 15
based 21
browser 17
```

[Read 209 lines]

The finalresults.txt file line totals matches up with that on my local machine as well so I can confidently say there should be 209 lines for the following versions of the program as well.

finalreport datap.txt - summary file with top 10% of most used words listed

```

546 //Create word frequency histogram
547 cout << "Start of merge thread" << endl;
548 //The first memory segment to read is the words from the first two threads
549 ifstream outfile;
550 ifstream infile;
551 //Push the results on each line of the results from the previous two threads, and stored in the results vector
552 string word;
553 vector<string> results;
554 //Open the first memory segment to be written to
555 outfile.open("finalreport_data.txt");
556 //Open the second memory segment and read words from the first threads results
557 infile.open("results_data1.txt");
558 //Read until end of file, first threads results
559 while (getline(infile, word))
560 {
561     //Push each word to the results vector
562     results.push_back(word);
563 }
564 //Close the first half of the words buffer
565 infile.close();
566 //Open the second half of the filtered words from the second threads results
567 infile.open("results_data2.txt");
568 //Read until end of file, second threads results
569 while (getline(infile, word))
570 {
571     //Push each word to the results vector
572     results.push_back(word);
573 }
574 //Create a map of a string and an integer value
575 //Open string buffer, the word and the integer is the frequency
576 map<string, int> M;
577 //Iterate through each word in the vector of all words
578 for (int i=0; i<results.size(); i++)
579 {
580     M[results[i]] += M[i];
581     // If the word is only found once it is end of map then the count is one
582     if (results[i] == i)
583     {
584         M[results[i]] = 1;
585     }
586     //Increment the count of the frequency in the map
587     M[results[i]]++;
588 }
589 //Close is the total amount of words found
590 int size = results.size();
591 //Open a file to store all the words found
592 int temp = ceil(size*.1);

```

```

1 //for each iterator to go through each word in the map
2 for(auto it=it1;
3 {
4     //if the frequency of the word is more than 10 and it is the final report summary
5     if((it.second > 10)
6         and(it < it1.first < " " && it < it2.second < " \n");
7     {
8         //Calculate word and thread
9         cout << "End of merge thread" << endl;
10         pthread_exit(NULL);
11     }
12 }
13
14 //this function that deals with creating and joining the threads as well as determining time to complete program. (4)
15 int main()
16 {
17     //intended to hold the variables in order to find completion time
18     struct timeval start, end;
19     gettimeofday(&start, NULL);
20
21     pthread_t id[MEM_THRESHOLD];
22
23     //creating the first two threads to deal with the first two arrays
24     cout << "Creating the first thread that calls firstThread() << endl;
25     pthread_create(&id[0], NULL, firstThread, (void *)0);
26     cout << "Creating the second thread that calls secondThread() << endl;
27     pthread_create(&id[1], NULL, secondThread, (void *)0);
28
29     //waiting for the first two threads to complete
30     for(int i=0; i<2; i++)
31     {
32         pthread_join(id[i], NULL);
33     }
34
35     //creating a thread to merge the results of the previous two threads
36     cout << "Creating a third thread to merge results" << endl;
37     pthread_create(&id[2], NULL, merge, (void *)0);
38     //complete the merge thread
39     pthread_join(id[2], NULL);
40
41     //intended for the program to complete and time will now be calculated
42     cout << "Process is complete" << endl;
43
44     //let the time the program finishes at
45     gettimeofday(&end, NULL);
46
47     //get micro and end time and take for the program to complete and print out the results (3)
48     int micro = end.tv_usec - 1000000 + end.tv_usec;
49     int micro_end = start.tv_usec + 1000000 + start.tv_usec;
50     cout << "Total time for data parallel execution: " << (micro_end - micro_start) << " microseconds" << endl;
51
52     return 0;
53 }

```

```

363  /** Sources - Project 1 Data Parallel Implementation
364  * (1) Geeks for Geeks
365  * - https://www.geeksforgeeks.org/program-to-find-frequency-of-each-element-in-a-vector-using-map-in-c/
366  * -- I used this resource to implement a way to find the frequency of each word in the vector. This example
367  * -- uses integers instead of strings, but I was able to utilize it to change the map variable to string.
368  *
369  * (2) Canvas Project 1 Discussion Board
370  * -- I was able to utilize the code posted on the Project 1 discussion board thanks to Dr. Bigriggs post.
371  * -- The majority of the text analytic process in the main function is from the post.
372  *
373  * (3) Canvas Lecture Notes
374  * - Bsharedmem.ppt Slide 7
375  * -- I was able to utilize timing for my program by using the lecture notes as an example.
376  * -- A struct is used and then the start and end times are referenced and used for calculations later on.
377  *
378  * (4) Tutorialspoint
379  * - https://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm
380  * -- Used as a reference for the syntax of threads and how to create/join
381  */

```

The screenshots on the page above are for the data parallel implementation of the project. The code may also be viewed in the submitted files as dataParallel.cpp. Zoom in for a better view of the code or open the specified file.

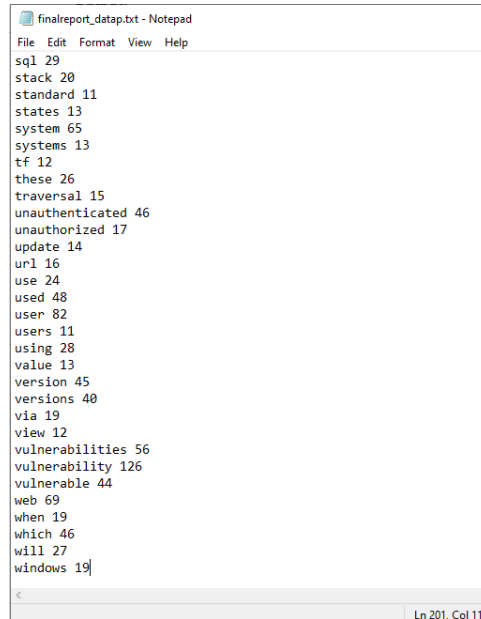
Output:

From my local machine:

```

C:\Users\perih\Downloads\ECE1570_Project1\ECE1570_Project1>dataParallel.exe
Creating the first thread that calls firstThread
Creating the second thread that calls secondThread
Start of the first thread
Start of second thread
End of first thread
End of second thread
Creating a third thread to merge results
Start of merge thread
End of merge thread
Process is complete
Total time for data parallel execution: 179443 microseconds

```



finalreport_datap.txt - Notepad

File Edit Format View Help

```

sql 29
stack 20
standard 11
states 13
system 65
systems 13
tf 12
these 26
traversal 15
unauthenticated 46
unauthorized 17
update 14
url 16
use 24
used 48
user 82
users 11
using 28
value 13
version 45
versions 40
via 19
view 12
vulnerabilities 56
vulnerability 126
vulnerable 44
web 69
when 19
which 46
will 27
windows 19

```

Ln 201, Col 11

The execution of the program took 179443 microseconds on my local machine. In the text file, it specifies there are 201 lines meaning that the top 10% of most used words summary file listed 201 different words. The results are different from the serial version meaning there is some type of error related when introducing threading.

Bigrigg Server:

```
[northampton]$ ./dataParallel
Creating the first thread that calls firstThread
Creating the second thread that calls secondThread
Start of the first thread
Start of second thread
End of first thread
End of second thread
Creating a third thread to merge results
Start of merge thread
End of merge thread
Process is complete
Total time for data parallel execution: 72279 microseconds
```

From the ssh server bigrigg.com, the execution took 72279 microseconds which is also significantly less than that of my local machine similar to what happened in the serial implementation.

```
able 78
access 73
additional 23
admin 11
administrative 19
adobe 11
advisory 20
affected 31
algorithm 13
all 12
allow 40
allows 25
also 20
any 15
application 56
applications 15
arbitrary 72
attack 35
attacker 169
attacks 13
authenticated 19
authentication 25
available 15
based 21
browser 17
```

[Read 209 lines

The finalresult_datap.txt file line totals matches up with that on my local machine serial implementation. I am still unsure as to why the results are different when checking from the server and from my local machine. If I had to say, I would think it would be due to the fact of having multiple cores on the server as well as it being a shared machine.

finalreport_taskP.txt - summary file with top 10% of most used words listed

[illegible]


```

150 //Join the threads
151 for(int i=0; i<i1; i++)
152 {
153     pthread_join(ids[i], NULL);
154 }
155
156 //Create a fourth thread to analyze the filtered words and join the thread
157 cout << "Creating the fourth thread that calls frequencyToFileThread" << endl;
158 pthread_create(&ids[3], NULL, frequencyToFileThread, (void *)0);
159 pthread_join(ids[3], NULL);
160
161 //Close both files
162 finalOutFile.close();
163 wordsOutFile.close();
164
165 //Indicate the program has complete
166 cout << "End of main program" << endl;
167
168 //Get the time the program finishes at
169 gettimeofday(&end, NULL);
170
171 //Calculate the time that was taken for the program to complete and print out the results (3)
172 (int) micro_end = end.tv_sec * 1000000 + end.tv_usec;
173 (int) micro_start = start.tv_sec * 1000000 + start.tv_usec;
174 cout << "Total time for task parallel execution: " << (micro_end - micro_start) << " microseconds" << endl;
175
176 return 0;
177 }
178
179 /** Sources - Project 1 Task Parallel Implementation
180 * (1) Tools for notes
181 * - https://www.gettyimages.com/program-to-find-frequency-of-each-element-in-a-vector-using-map-in-c/
182 * - I used this resource to implement a way to find the frequency of each word in the vector. His example
183 * - uses integers instead of strings, but I was able to utilize it to change the map variable to string.
184 *
185 * (2) Canvas Project 1 Discussion Board
186 * - I was able to utilize the code posted on the Project 1 discussion board thanks to Dr. Algrigg's post.
187 * - The majority of the test analysis process in the main function is from the post.
188 *
189 * (3) Canvas Lecture Notes
190 * - Behavemem.ppt Slide 7
191 * - I was able to utilize slides for my program by using the lecture notes as an example.
192 * - A struct is used and then the start and end times are referenced and used for calculations later on.
193 *
194 * (4) Tutorialspoint
195 * - https://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm
196 * - Used as a reference for the syntax of threads and how to create/join
197 *
198 * (5) Oracle Mutexes
199 * - https://docs.oracle.com/cd/E13455-01/800-529/comp-12/index.html
200 * - Used as a reference for the syntax of mutexes and how to use regarding locking and unlocking
201 */

```

The screenshots on the page above are for the task parallel implementation of the project. The code may also be viewed in the submitted files as taskParallel.cpp. Zoom in for a better view of the code or open the specified file.

Output:

From my local machine:

```

C:\Users\perih\Downloads\ECE1570_Project1\ECE1570_Project1>taskParallel.exe
Start of main program
Creating the first thread that calls readInDataThread
Creating the second thread that calls formatStringThread
Start of readInDataThread
Locking wordsFromStringVectorMutex
Creating the third thread that calls listWordsStringtoVectThread
Joining all threads
Unlocking wordsFromStringVectorMutex
End of readInDataThread1
Start of formatStringThread
Locking wordsFromStringVectorMutex and filteredWordsVectorMutex
Unlocking wordsFromStringVectorMutex and filteredWordsVectorMutex
End of formatStringThread
Start of listWordsStringtoVectThread
Locking filteredWordsVectorMutex and wordsOutFileMutex
Unlocking filteredWordsVectorMutex and wordsOutFileMutex
End of listWordsStringtoVectThread
Creating the fourth thread that calls frequencyToFileThread
Start of frequencyToFileThread
Locking wordsOutFileMutex and finalOutFileMutex
Unlocking wordsOutFileMutex and finalOutFileMutex
End of frequencyToFileThread
End of main program
Total time for task parallel execution: 888178 microseconds

```

The execution of the program took 888178 microseconds on my local machine. In the text file, it specifies there are 209 lines. The results are the same as the serial version meaning there is consistency between the two implementations.

finalReport_taskP.txt - Notepad

File Edit Format View Help

```

unauthenticated 46
unauthorized 17
update 14
url 16
use 24
used 48
user 82
users 11
using 28
value 13
version 45
versions 40
via 19
view 12
vulnerabilities 56
vulnerability 126
vulnerable 44
web 69
when 19
which 46
will 27
windows 19
wireless 14
within 20
without 14
www 12
x 148
xml 26
xss 11
zip 31

```

<

Ln 209, Col 7

Bigrigg Server:

```
[northampton]$ ./taskParallel
Start of main program
Creating the first thread that calls readInDataThread
Creating the second thread that calls formatStringThread
Start of readInDataThread
Locking wordsFromStringVectorMutex
Creating the third thread that calls listWordsStringtoVectThread
Joining all threads
Unlocking wordsFromStringVectorMutexStart of formatStringThread
End of readInDataThread
Locking wordsFromStringVectorMutex and filteredWordsVectorMutex
1
Unlocking wordsFromStringVectorMutex and filteredWordsVectorMutexStart of listWordsStringtoVectThread
End of formatStringThread

Locking filteredWordsVectorMutex and wordsOutFileMutex
Unlocking filteredWordsVectorMutex and wordsOutFileMutex
End of listWordsStringtoVectThread
Creating the fourth thread that calls frequencyToFileThread
Start of frequencyToFileThread
Locking wordsOutFileMutex and finalOutFileMutex
Unlocking wordsOutFileMutex and finalOutFileMutex
End of frequencyToFileThread
End of main program
Total time for task parallel execution: 649249 microseconds
```

From the ssh server bigrigg.com, the execution took 649240 microseconds which is less than that of my local machine similar to what happened in the serial implementation, but the difference is not as drastic.

```
able 78
access 73
additional 23
admin 11
administrative 19
adobe 11
advisory 20
affected 31
algorithm 13
all 12
allow 40
allows 25
also 20
any 15
application 56
applications 15
arbitrary 72
attack 35
attacker 169
attacks 13
authenticated 19
authentication 25
available 15
based 21
browser 17
```

[Read 209 lines]

The finalresult_taskp.txt file line totals matches up with that on my local machine serial implementation which is 209 lines. The results from my local machine and from the bigrigg server are the same amount of lines, just differing execution times.

Optimizations

There are multiple optimizations that could be made to each implementation, but I would like to address some of the more obvious optimizations that would make a significant difference in the implementation and possibly efficiency of the program.

Starting off specifically with the serial implementation, the data files could be read in from the directory using a command instead of being hardcoded into an array. This could also be similar when regarding the noise words, they could be stored in a text file, read in, and then eliminated from the main data if found. This would be better for cleaner code and less lines as well, possibly less need for loops to iterate through the array rather than just finding the files from the directory itself and directly accessing the data. The program would be able to recursively search through the files and this implementation could be applied to each part of the project.

Another large optimization that could be made is to create more threads in the task parallel version to be more of a streamlined process being able to break up the tasks even more. Some of my threads have long processes that would take awhile or need to wait for a previous thread to finish where as if they were broken up into smaller parts there would be less waiting time and more tasks would be able to run at the same time.

Finally, the last optimization I wanted to mention was to be able to pass data into a thread from the main thread or main method. I had the code set up, but to speed up my understanding of the parallel programming portion of the project, I decided to hardcode some information in the threads or make the variables global. In the future I would plan to pass these variables into the thread so as not to cause dangerous access to such global variables.