Project 4 Write Up

Hunter Guthrie

Calvin Bailey

Payton Jellison

Architecture

Both of the programs share the same three global arrays and two defines denoted below.

- char wiki_entry
  - This array holds the entry wiki file.
  - It is a 2D array with dimensions of 1,000,000 and 2003
- int LCSuf
  - This array is used to hold information about the two strings that are being compared in the “findThings” function. Declaring it inside of the findThings function segfaults.
  - It is a 2D array with dimensions of 2003 and 2003
- char results_array
  - This array holds the largest common substring between 2 strings.
  - It is a 2D array with dimensions of 1,000,000 and 2003
- Int thread_amount
  - This int is only used in pthread. It holds the number of threads the user wants the program to run
- #define array_size 1000000
  - This define is used to denote the size of the wiki file. It is used throughout the program.
- #define wiki_string 2003
  - This define is used to denote the size of the wiki file lines. It is used throughout the program.

This program is split up into five functions. These function, as well as their tasks are outlined as follows:

- Int main()
    - This is the main function of the program. This function handles the gathering of the performance metrics as well as makes any function calls that are needed.
- Int wiki_to_array()
    - This function handles reading in the file and stores it in the global “wiki_entry” array.
- Char * LargestCommonSubstring()
    - This function deals with the preliminary tasks needed before the largest common substring can be found. Once these tasks have been completed, it will call the “findThings” function function. This is the main function that is multithreaded.
- Int findThings()
    - This function is used to find the largest common substring from two strings.
    - There are four parameters.
        - string1
            - REQUIRED
            - A string that is being compared
        - string2
            - REQUIRED
            - A string that is being compared
        - m
            - REQUIRED
            - Length of string1
        - n
            - REQUIRED
            - Length of string2
    - Once a substring has been found, it is placed inside the global “results_array” array to be printed later.
- Int print()
    - This function displays the “results_array” to the user.

Run time

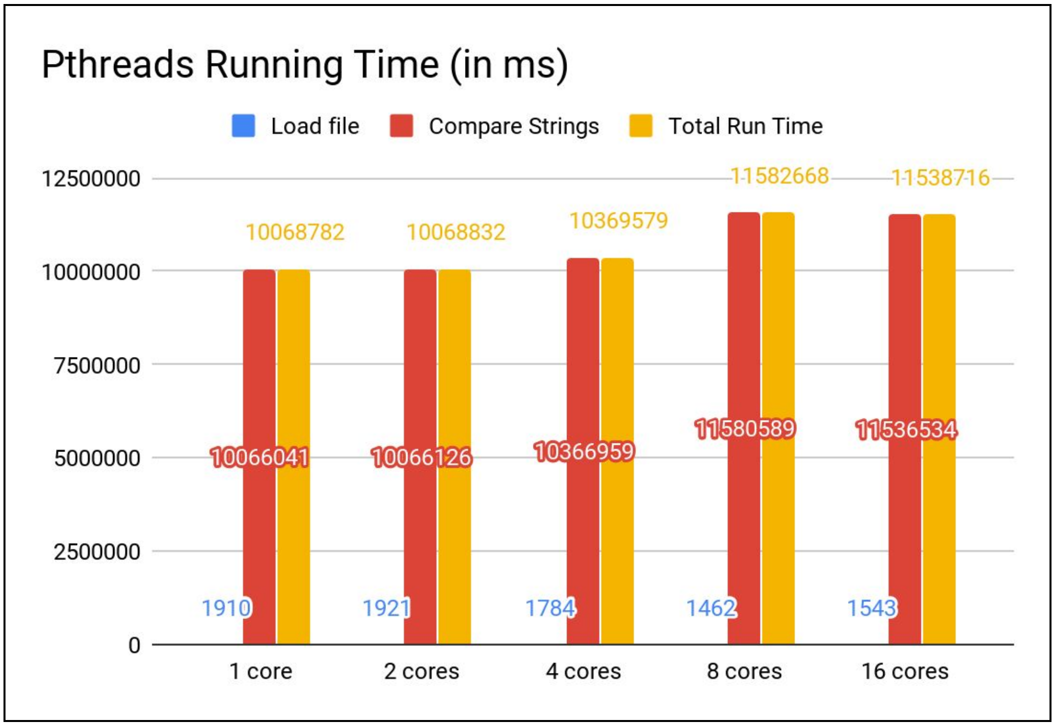
In order to measure the runtime of this program, the “sys/time.h” library was used. Five timeval structs were created to keep track of the time.The function “gettimeofday(&<timeval_struct>, NULL);” allows the ability to set the timeval structs to the current time. Using this function before and after a section of code and subtracting the difference gave us the amount of time that the section took to run in milliseconds (ms). This was used to find the run time of the following sections:

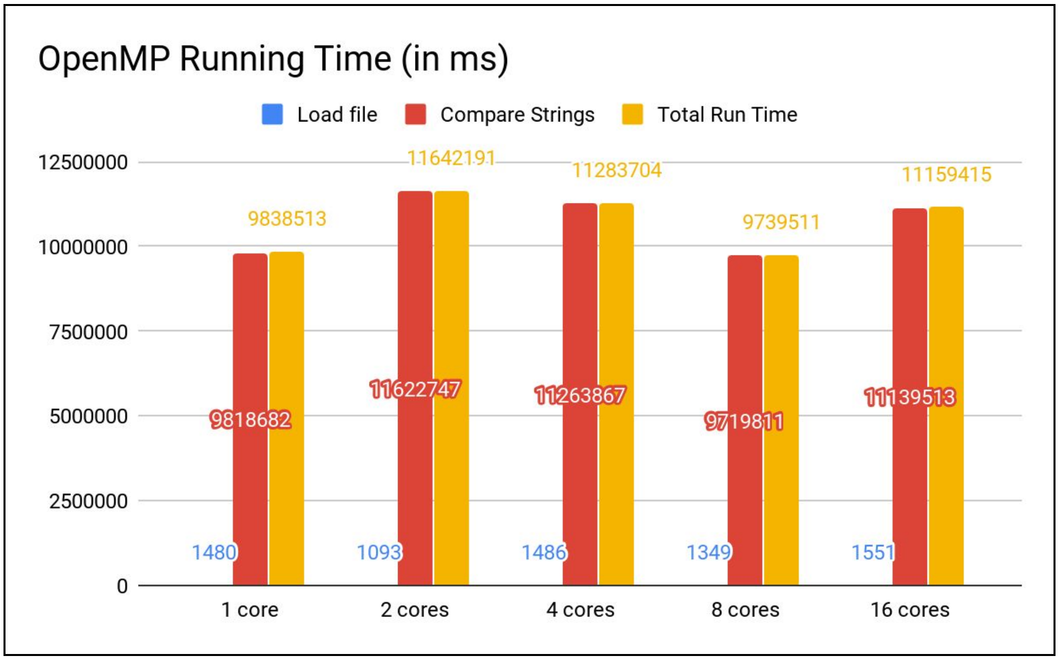
- Time to load the file
- Time to compare all of the strings
- Total system run time

Unfortunately, the times for the string comparison in openMP reported times that were incorrect. This problem was resolved by calculating an approximated time instead. The following steps were used:

1. Ran the “print” function to find how long it takes to print
    a. This test was ran five times and then averaged
    b. The average yielded 18350.6 which was rounded to 18351
2. Subtracted the total time from the print and the approximated load file time

Next, each program was ran five times with a different number of cores (1,2,4,8,16) per run. The times for these runs are located in the graphs below. The results indicate that openMP is the fastest as well as the slowest multithreaded solution. The fastest run was with 8 cores with a total run time of 2.7 hours and 2 cores being the slowest with a total run time of 3.23 hours. Pthreads was consistently taking longer then 2.7 hours to complete. During small scale testing, which dealt with a smaller text file, the run times increased more between the amount of cores then with the large scale runs.


Pthreads Running Time (in ms)
Load file
Compare Strings
Total Run Time
12500000
10000000
7500000
5000000
2500000
0
11582668
11538716
10068782
10068832
10369579
10066041
10066126
10366959
11580589
11536534
1910
1921
1784
1462
1543
1 core
2 cores
4 cores
8 cores
16 cores

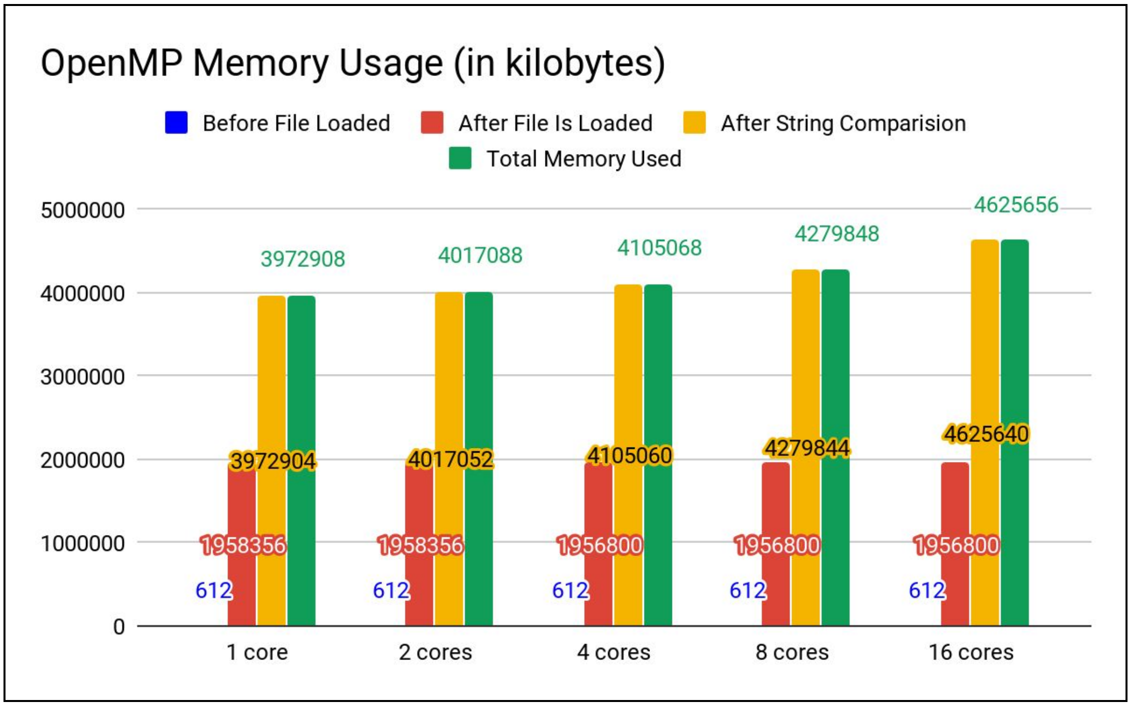

OpenMP Running Time (in ms)
Load file
Compare Strings
Total Run Time
12500000
10000000
7500000
5000000
2500000
0
11642191
11283704
11159415
9838513
9739511
9818682
11622747
11263867
9719811
11139513
1480
1093
1486
1349
1551
1 core
2 cores
4 cores
8 cores
16 cores

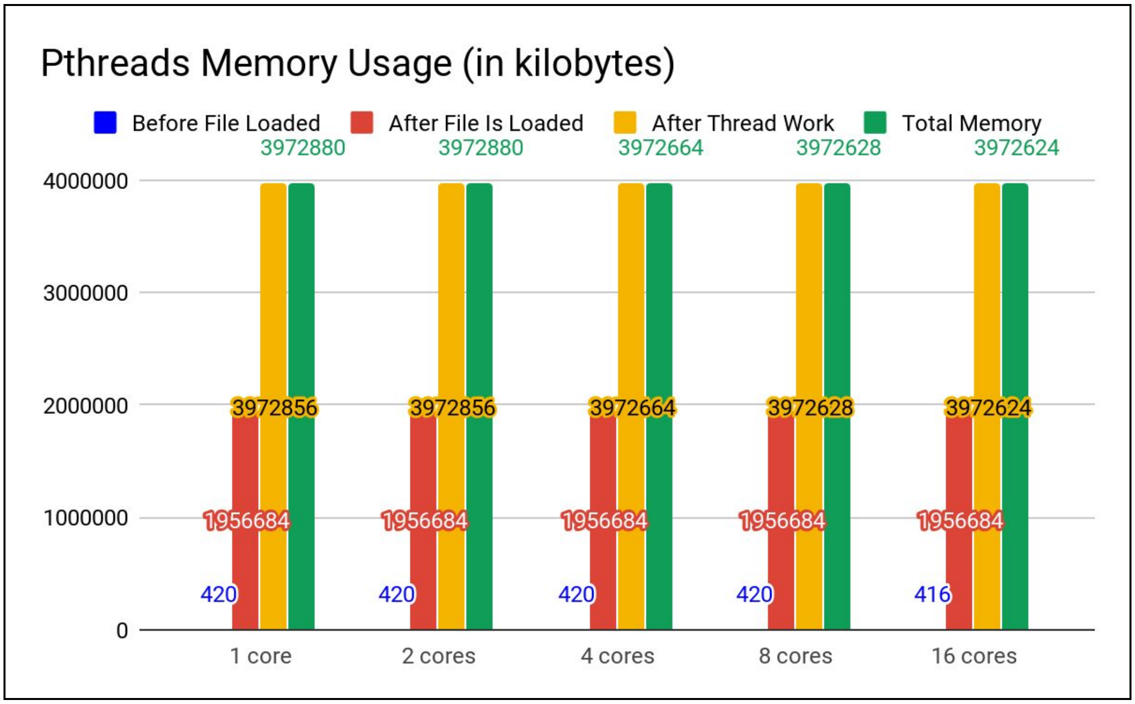
Memory usage

Memory usage was tracked with the use of the “getrusage()” function. This function was used at the following locations:

- Before wiki was loaded
- After wiki was loaded
    a. This is also before the string comparisons were made
- After the string comparisons were done
    a. This is also before the “print” function
- After the “results_array” was printed
    a. This is also the end of the program

This function allowed us to see how much memory the program was using at different stages. The graphs below show what the results of these measurements were for both openMP and pthreads. The graphs show the memory usage in kilobytes to give a more detailed description of memory usage at each stage. The results indicate that pthreads uses much less memory on average than openMP no matter the core count. OpenMP memory usage consistently increased when it was given more cores. This could mean that pthreads is overall more memory optimized then openMP. Alternatively, because of how close the run times are between each other, it may not be properly implemented.


OpenMP Memory Usage (in kilobytes)


Pthreads Memory Usage (in kilobytes)

Problems

- We believe that the program is leaking memory due to us not being able to free a variable called “char resultStr”. Everytime we would try to free the variable, it would segfault the program. Leaving out the free allows the program to run.

- The array “LCSuf” that is being used inside the “findThings” function is globally defined. When trying to constrain it to just the “findThings” function, it segfaults whenever the size is bigger then 1000x1000. To fix the issue, it was moved to a global position.

- If we run pthreads with more than 1 core we are not able to get any output, so our multithreading is done through threads not cores.

- We also have a problem where some of the largest common substrings are not created. We believe this is due to race conditions. The race conditions are occuring in the nested for loops that fill the array named “LCSuf”. We tried placing locks where data was being added to the array, as well as where the function was being called. Neither method fixed the problem. We did find that placing a lock around the for loops worked, but this caused it to become single threaded so it was removed.

- Looking at the results we received, we believe that we may not have implemented pthreads and openMP correctly as we were expecting faster run times when more cores were added. And while the general trend did suggest this, the numbers seem too close together.