**Part 1**

1. **Explain the problem and identify evaluation metrics for experiments**

Is multithreading a faster alternative to sequential operation when finding the number of substrings in a given string, S1? This will be conducted using a sequential program as well as an altered multithreaded program with 3 different inputs for each. Recorded results will be in milliseconds.

1. **Explain your choice of threading libraries**

I chose to use pthreads as I have not done much threading in the past. Therefore, I had no bias for another library or interest in trying anything new, since this was my first time using pthreads, which was the recommended library.

1. **Explain the design of the experiment and develop programs for evaluation**

The programs used are the given substring.c and an adapted version called substring\_thread.c.

substring.c loops through a given string of N1 characters checking the substring at that spot and beyond from that spot, x, to x+N2 (number of characters in given substring, S2).

substring\_thread.c works the same way as substring.c, except the string S1 is split into an equal number of threads to run simultaneously, hopefully speeding up the process. The number of threads used will satisfy the given conditions (Let T be the number of threads):

* N1%T == 0
* N1/T > N2
* T < # Cores in VM (If too many threads, then multithreading is inviable).

**Experiment:**

1. Run the first test file in substring.c and collect the time it took to complete.
2. Run the first test file in substring\_thread.c and collect the time it took to complete.
3. Repeat steps 1-2 four more times to fill in the data table.
4. Repeat steps 1-3 two more times with the following 2 files and subsequent data tables.
5. **Detail your collected experimental results**

|  |  |  |
| --- | --- | --- |
| **test1.txt**  **(N1 = 21, N2 = 2, Threads = 3)** | | |
| **Run** | **Sequential (ms)** | **Multithreaded (ms)** |
| **1** | 0.501 | 0.703 |
| **2** | 0.500 | 0.643 |
| **3** | 0.501 | 0.725 |
| **4** | 0.500 | 0.761 |
| **5** | 0.500 | 0.655 |
| **Avg** | 0.500 | 0.697 |

|  |  |  |
| --- | --- | --- |
| **hamlet.txt**  **(N1 = 142,954, N2 = 6, Threads = 2)** | | |
| **Run** | **Sequential (ms)** | **Multithreaded (ms)** |
| **1** | 0.826 | 0.861 |
| **2** | 0.903 | 0.830 |
| **3** | 0.818 | 0.814 |
| **4** | 0.814 | 1.002 |
| **5** | 0.828 | 0.819 |
| **Avg** | 0.838 | 0.865 |

|  |  |  |
| --- | --- | --- |
| **shakespeare.txt**  **(N1 = 4,358,846, N2 = 3, Threads = 2)** | | |
| **Run** | **Sequential (ms)** | **Multithreaded (ms)** |
| **1** | 15.548 | 8.555 |
| **2** | 15.691 | 10.205 |
| **3** | 15.550 | 8.356 |
| **4** | 15.347 | 8.353 |
| **5** | 15.489 | 8.588 |
| **Avg** | 15.525 | 8.811 |

1. **Analyze, graph, and interpret experimental results and draw conclusions**

**Analyze, interpret, and conclude**

From the data I collected, it appears that multithreading is not always the better option when considering the speed of completion for a given program. test1.txt completes faster when run by a sequential program, hamlet.txt could be considered a negligible difference in average completion times between sequential and multithreaded, and shakespeare.txt completes a significant amount faster on a multithreaded program. The difference in performance between a sequential and multithreaded program for varying lengths of strings, S1, could be related to the amount of time it takes to create a number of threads. When S1 is short, then the amount of time used to create threads and join them back together is not worth it. However, when S1 is very long, then the amount of time used to create threads and rejoin them is negligible compared to the time a sequential program takes to go through S1 “alone.” For this given case, multithreading would be a better route if S1 is very large. However, a sequential program would be better suited for shorter variants of S1.

**Part 2**

1. **Explain the problem and identify evaluation metrics for experiments**

How does the length of a given input string effect the speed of completion for the producer-consumer algorithm? Time will be recorded in milliseconds and will be recorded as an average of 5 runs for 3 different files.

1. **Explain your choice of threading libraries**

I chose to use pthreads as I got somewhat familiar with them in Part 1 of this assignment. However, I had not used semaphores with them, so I chose to build upon the knowledge I gained rather than start fresh with a new library.

1. **Explain the design of the experiment and develop programs for evaluation**

**Experiment:**

1. Run procon.c with “message.txt” as input message. Record time for 5 total runs and then average all 5.
2. Repeat step 1 with “message2.txt” and “message3.txt”.
3. Graph the 3 averages on 1 graph to determine linearity.
4. **Detail your collected experimental results**

|  |  |  |  |
| --- | --- | --- | --- |
| **Filename** | **Message.txt** | **Message2.txt** | **Message3.txt** |
| **String Length** | **78** | **448** | **1622** |
| **Run 1 (ms)** | 4.533 | 8.710 | 30.314 |
| **Run 2 (ms)** | 4.549 | 8.628 | 40.501 |
| **Run 3 (ms)** | 3.835 | 10.494 | 30.634 |
| **Run 4 (ms)** | 4.527 | 7.071 | 28.751 |
| **Run 5 (ms)** | 4.492 | 7.250 | 24.097 |
| **Avg** | **4.387** | **8.431** | **30.859** |

1. **Analyze, graph, and interpret experimental results and draw conclusions**

**Analyze, interpret, and conclude**

It appears that completion time is dependent on string length in some respects. As string length increases, so does the time to completion. The trendline equation fits the values with acceptable error. This error is likely primarily due to factors of my own machine rather than the algorithm, as all runs were done on 1 VM. However, the proportionality of length and time make sense since the longer a string is, the longer it takes for the program to process it. In my producer-consumer algorithm, string length and completion time are proportionally related to one another. Thus, making them linear.

**Part 3**

1. **Explain the problem and identify evaluation metrics for experiments**
2. **Explain your choice of threading libraries**
3. **Explain the design of the experiment and develop programs for evaluation**
4. **Detail your collected experimental results**
5. **Analyze, graph, and interpret experimental results and draw conclusions**