**CSC 4420 Lab 4**

**Objectives:**  
1.) Using the provided code at https://www.geeksforgeeks.org/merge-sort-using-multi-threading/ , rewrite the code to test the time it takes to process 10,000,000 elements with 1,2,3,8, and 16 threads respectively.

2.)Provide a graph of threads vs. time

3.)Provide the code used.

**Results:**Code Structure:  
I really don’t like turning in multiple pieces of code, because it means that if I make a change to the base template, I have to make a change to EVERYTHING. The key differences in my code are the following:  
1.) I defined a mutual exclusion variable

2.)I defined a thread merge function which slices the array into equal sizes based upon the thread assigned, that particular thread then only works on that portion of the array. The value of what is being sliced is locked by the mutual exchange variable and incremented so each thread works on a different part.

3.) I generalized the main function into a function that allows input for any given number of threads purely so I only had to turn in one piece of code.

ISSUE: The code that works in an online compiler does NOT work in the provided Linux operating system that is included with VMWareHorizonClient. This means the results may be wonky as they cannot be properly tested on my computer. To try to offset the timing issues that may be associated with serverside calculations, I ran the program multiple times so I could create a trend line instead.

|  |  |  |
| --- | --- | --- |
| **Number of Elements:** | **Number of Threads:** | **Time (seconds):** |
| 1,000,000 | 1 | **0.10635** |
|  | 1 | **0.16574** |
|  | 1 | **0.166282** |
|  | 1 | **0.169754** |
|  | 1 | **0.226652** |
|  | 1 | **0.225725** |
|  | 1 | **0.166668** |
| 1,000,000 | 2 | **0.10483** |
|  | 2 | **0.182024** |
|  | 2 | **0.169221** |
|  | 2 | **0.167867** |
|  | 2 | **0.30016** |
|  | 2 | **0.238349** |
|  | 2 | **0.181957** |
|  | 2 | **0.169229** |
|  | 2 | **0.308355** |
| 1,000,000 | 4 | **0.096339** |
|  | 4 | **0.225803** |
|  | 4 | **0.237137** |
|  | 4 | **0.224324** |
|  | 4 | **0.221075** |
|  | 4 | **0.23376** |
|  | 4 | **0.362284** |
|  | 4 | **0.259725** |
|  | 4 | **0.195728** |
| 1,000,000 | 8 | **0.096042** |
|  | 8 | **0.294465** |
|  | 8 | **0.356424** |
|  | 8 | **0.255147** |
|  | 8 | **0.27467** |
|  | 8 | **0.361546** |
|  | 8 | **0.239048** |
|  | 8 | **0.271712** |
|  | 8 | **0.359157** |
| 1,000,000 | 16 | **0.092298** |
|  | 16 | **0.346438** |
|  | 16 | **0.357877** |
|  | 16 | **0.393895** |
|  | 16 | **0.257606** |
|  | 16 | **0.288465** |
|  | 16 | **0.285344** |
|  | 16 | **0.246533** |
|  | 16 | **0.362429** |

Keep in mind that the following results where the amount of time increases for the number of threads is to be expected if you look at the structure of the code, even though the threads sort each initial slice of the main array, the main array still needs to be merged without using threads. This means the more slices performed, the more merging must actually occur.   
  
Please note professor that I am unable to run the code on the provided linux machine, for reasons unknown it does not take kindly to threading.   
  
A graph with blue dots and a line

Description automatically generated