**Student Names**

Trung Hoang

Cortney Reed

**Assignment Discussions**

1. Describe the parallelization process, using code or pseudocode to help the discussion.

Version 1

The first and easier version of this parallelization problem is to create different worker threads for division and elimination steps. These two worker threads will be called for each value of k. Although this parallelization is faster than serial version, it is still slow because of the overhead of forking and joining threads for every value of k.





The following table show the execution time for the version 1 of the parallelization.

Table 1. Execution time of parallelization (version 1) vs serialized version

|  |  |  |  |
| --- | --- | --- | --- |
| **No of elements** | **No of threads** | **Execution Time (s)** | |
| **Multithreaded** | **Serial Version** |
| **512** | **4** | 0.21 | 0.07 |
| **8** | 0.51 |
| **16** | 1.05 |
| **32** | 2.09 |
| **1024** | **4** | 0.43 | 0.32 |
| **8** | 0.98 |
| **16** | 2.07 |
| **32** | 4.22 |
| **2048** | **4** | 2.15 | 3.74 |
| **8** | 3.07 |
| **16** | 4.73 |
| **32** | 13.59 |
| **4096** | **4** | 14.88 | 38.58 |
| **8** | 15.07 |
| **16** | 21.52 |
| **32** | 26.52 |

Version 2

In order to achieve better performance for the gaussian elimination program, we used barrier synchronization and pthreads. The idea to have each worker thread perform both division and elimination steps. However, there is a barrier between the steps to make sure all threads finish the division step before moving on. The reason for this is because elimination thread depends on the data from the divisions step. The following pseudocode shows how worker thread handle the task. It is important to point out that there is a second barrier use at the end of the loop to make sure the elimination step is finished before continuing the iteration.



Because the worker thread now handles all the work, the main program only needs to fork and join the threads.



The following table show the improvement of the execution time for different numbers of elements and threads. This is significantly better than version 1.

Table 2 Execution time of parallelization (version 2) vs serialized version

|  |  |  |  |
| --- | --- | --- | --- |
| **No of elements** | **No of threads** | **Execution Time (s)** | |
| **Multithreaded** | **Serial Version** |
| **512** | **4** | 0.06 | 0.06 |
| **8** | 0.05 |
| **16** | 0.10 |
| **32** | 0.17 |
| **1024** | **4** | 0.25 | 0.32 |
| **8** | 0.20 |
| **16** | 0.30 |
| **32** | 0.40 |
| **2048** | **4** | 1.65 | 4.02 |
| **8** | 1.18 |
| **16** | 1.11 |
| **32** | 1.44 |
| **4096** | **4** | 14.17 | 28.38 |
| **8** | 13.11 |
| **16** | 13.15 |
| **32** | 11.59 |