**Student Names**

Trung Hoang

Cortney Reed

**Assignment Discussions**

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

Version 1

In order to implement a parallelized version of the Jacobi solver program provided we used barrier synchronization and pthreads. This problem can be broken down into 2 main steps. The first step is calculating new\_x. The second step is calculating termination conditions (mean square error, number of iterations) and assign new\_x to x. The second step depends on the first step. The way we did this parallelization is to have the worker threads perform the first step and the main program perform the second step. We use barriers to communicate between main and worker threads.



It is noted that “done” is shared between main() and worker threads. By doing it this way, worker threads will know when to terminate. The following pseudocode show how the worker thread handles the job.



Table 1 Execution time of version 1 vs serialized version

|  |  |  |  |
| --- | --- | --- | --- |
| **No of elements** | **No of threads** | **Execution Time (s)** | |
| **Multithreaded** | **Serial Version** |
| **512** | **4** | 0.67 | 3.77 |
| **8** | 0.71 |
| **16** | 1.16 |
| **32** | 1.39 |
| **1024** | **4** | 4.73 | 29.36 |
| **8** | 2.35 |
| **16** | 2.98 |
| **32** | 4.61 |
| **2048** | **4** | 32.06 | 258.55 |
| **8** | 17.98 |
| **16** | 12.28 |
| **32** | 13.72 |

To improve version 1, we decide to apply parallelization to the second step as well. This is done by having the worker threads calculating the partial sum then sum them up to a variable which is locked by mutex. On top of that, we also apply “ping pong” to x and new\_x. With this, main() only has the function of forking and joining the threads as follows:



This will continue until convergence. Below is the pseudocode of the worker threads



Table 2 Execution time of version 2 vs serialized version

|  |  |  |  |
| --- | --- | --- | --- |
| **No of elements** | **No of threads** | **Execution Time (s)** | |
| **Multithreaded** | **Serial Version** |
| **512** | **4** | 0.80 | 3.77 |
| **8** | 0.66 |
| **16** | 0.87 |
| **32** | 1.51 |
| **1024** | **4** | 4.10 | 29.36 |
| **8** | 2.74 |
| **16** | 2.40 |
| **32** | 3.42 |
| **2048** | **4** | 31.49 | 258.55 |
| **8** | 16.56 |
| **16** | 12.37 |
| **32** | 14.82 |

Although in theory, version 2 should be faster than version 1 but based on the result, it is similar in most cases. This is because the size of the matrix that need to compute in the second step is very small compared to the first step (n vs n^2). Therefore, optimization on this step but adding more overhead from mutex seems to cancel each other out and as the result, version 1 and version 2 are comparable.