**Lab 3 Report & Screenshots**

Linux OS (ECS machine)  
**Number of cores**: 4 (Applies for all tests)

**Scheme 0**: Sequential multiplication (single threaded)  
**Scheme 1**: Threaded multiplication with parent waiting for all children  
**Scheme 2**: Threaded multiplication with parent continually checking on children  
**Scheme 3**: Threaded multiplication with parent waiting on a semaphore

Test 1

|  |  |
| --- | --- |
| **Parameters:** Array size = 100M Threads = 2  Index for zero = 50M+1 | **Summary:** Schemes 0 and 1 have similar completion times because Scheme 1 must wait for all threads to complete, so while 1 thread finds the zero and exits early, it doesn’t matter because the other thread must continue execution. Schemes 2 and 3 are incredibly fast because the second thread for both schemes find the zero incredibly quickly, and therefore exit early since the array is split in half. |
| **Output screenshot:** | |

Test 2

|  |  |
| --- | --- |
| **Parameters:** Array size = 100M Threads = 4  Index for zero = 75M+1 | **Summary:** Unlike test 1, we see that Scheme 1 finishes execution before Scheme 0 because we have more threads traversing the array, even if they don’t exit early. We again see that Schemes 2 and 3 execute very quickly due to their ability to exit early and cancel all threads if a zero is found. The index for zero is 75M+1, meaning that if the array is split into 4 parts, one of the threads will almost immediately come across a zero and cancel all other threads for Schemes 2 and 3. |
| **Output screenshot:** | |

Test 3

|  |  |
| --- | --- |
| **Parameters:** Array size = 100M Threads = 8  Index for zero = 88M | **Summary:** Again, we see Schemes 2 and 3 with very fast performance due to their ability to cancel all threads if a zero gets found. Due to the added threads, we also see a faster completion time for Scheme 1 over Scheme 0. It is possible that we do not see a big jump in thread speed from 4 to 8 threads since the system only has 4 CPU cores, so some threads have to wait. |
| **Output screenshot:** | |

Test 4

|  |  |
| --- | --- |
| **Parameters:** Array size = 100M Threads = 2  Index for zero = -1  (no zero) | **Summary:** Now that there is no zero in the input array, times for Schemes 1, 2, and 3 are similar. Additionally, since Scheme 0 cannot exit early, we notice that the threaded Schemes are roughly twice as fast. |
| **Output screenshot:** | |

Test 5

|  |  |
| --- | --- |
| **Parameters:** Array size = 100M Threads = 4  Index for zero = -1  (no zero) | **Summary:** Again since we have no zero in the array, there is no early exit, and Schemes 1 and 3 are both roughly four times as fast. However, we can see that Scheme 2 is much slower than the 1 and 3. This could be because the busy waiting loop must continually loop and check all children. |
| **Output screenshot:** | |

Test 6

|  |  |
| --- | --- |
| **Parameters:** Array size = 100M Threads = 8  Index for zero = -1  (no zero) | **Summary:** We don’t see an eight times improvement with Schemes 1,2 and 3, which is likely due to the 8 threads having to wait and share the 4 CPU cores. Again, we see Scheme 2 being slower than 1 and 3 due to the busy waiting loop having to check and recheck all of the threads. |
| **Output screenshot:** | |

**Conclusion**

Scheme 0 with a single thread is the most inefficient because it is not able to distribute the work between multiple threads that work in parallel. Scheme 1 is also inefficient because with cases where there is a zero in the array, all threads cannot exit early, since all threads must be completed to exit. Even if one thread in Scheme 1 finds the zero early and exits early, all other threads must still finish calculating their products and terminate. Schemes 2 and 3 are the most efficient since they use some form of synchronization, busy waiting and semaphores respectively. However, there are times where Scheme 2, busy waiting, can be inefficient, such as with tests 5 and 6, which can be due to constantly checking each thread. This can lead to wasted CPU cycles and can lead to longer completion times. Overall, it seems Scheme 3, the semaphore solution, is the most efficient, since it does not have to check each thread’s status repeatedly because it just relies on a semaphore post. Threads have an up-to-date count of the completed threads, so there is no need to recheck the same thread like in Scheme 2. Furthermore, it can also exit early and cancel all threads, ensuring that it’s fast even if there is a zero in the array.