Assignment #2

Test#1 (mtfindmin 100M 2 50M+1)

1. Sequential search completed in 27 ms. Min = 0
2. Threaded FindMin with parent waiting for all children completed in 34ms. Min = 0
3. Threaded FindMin with parent continually checking on children completed in 0ms. Min =0
4. Threaded FindMin with parent waiting on a semaphore completed in 0ms. Min=0

Test#2 (mtfindmin 100M 4 75M+1)

1. Sequential search completed in 41ms. Min = 0
2. Threaded FindMin with parent waiting for all children completed in 17ms. Min = 0
3. Threaded FindMin with parent continually checking on children completed in 0ms. Min =0
4. Threaded FindMin with parent waiting on a semaphore completed in 1ms. Min=0

Test#3 (mtfindmin 100M 8 88M)

1. Sequential search completed in 48ms. Min = 0
2. Threaded FindMin with parent waiting for all children completed in 16ms. Min = 0
3. Threaded FindMin with parent continually checking on children completed in 15ms. Min =0
4. Threaded FindMin with parent waiting on a semaphore completed in 1ms. Min=0

Test#4 (mtfindmin 100M 2 -1)

1. Sequential search completed in 54ms. Min = 1
2. Threaded FindMin with parent waiting for all children completed in 34ms. Min=1
3. Threaded FindMin with parent continually checking on children completed in 34ms. Min=1
4. Threaded FindMin with parent waiting on a semaphore completed in 34ms. Min=1

Test#5 (mtfindmin 100M 4 -1)

1. Sequential search completed in 54ms. Min = 1
2. Threaded FindMin with parent waiting for all children completed in 18ms. Min=1
3. Threaded FindMin with parent continually checking on children completed in 30ms. Min=1
4. Threaded FindMin with parent waiting on a semaphore completed in 18ms. Min=1

Test#6 (mtfindmin 100M 8 -1)

1. Sequential search completed in 55ms. Min = 1
2. Threaded FindMin with parent waiting for all children completed in 16ms. Min=1
3. Threaded FindMin with parent continually checking on children completed in 26ms. Min=1

4. Threaded FindMin with parent waiting on a semaphore completed in 16ms. Min=1

Conclusion(with an 8 core machine) :

Sequential search takes more time to complete the farther out in the array you put the 0 (ex. 50M+1, 75M+1, 88M). In the cases where there were no zeroes, sequential search took the same amount of time regardless of thread size.

Parent waits for all children - The 2nd method total runtime was cut in half when the thread count went from 2 to 4 but going from 4 to 8 did not cut time in half because of the extra core required for exact parallel processing speed up. 1 core for the parent and 8 cores for the 8 children. When there was no zero, once it completed searching the last thread the volatile int comparison was triggered and we broke out of the loop and finished. This time was comparative to the parent waiting on a semaphore.

Parent checks on all children - The 3rd method was efficient until you needed more cores than the computer had. Once it required more cores, it would act more like the 1st method that waited for children to finish. For the last 3 tests when there were no zeroes in the array, it took longer than the other methods because it was spending more time competing for the CPU with the children. The reason that it did not seem like that much of a difference was when I tested it with 2 threads and no zero. This happened because there was not very many threads to compete with for CPU time, so it finished rather quickly.

Parent waits on a semaphore - The 4th method was the most efficient on most of the tests. When there was a zero, it found and exited faster than the 3rd method because it did not have to compete for CPU time. Once the parent found the 0 in the child it could exit immediately. When there wasn't a zero, it would complete the entire search of the last thread before it could finish, which would be the reason why it was the same time as the 2nd method of waiting for all the children to finish.