Performance Analysis

Over the next few pages, you will see the data that I collected for this experiment. I have to say, it is pretty interesting to see how different strategies affect the outcome of the performance of an algorithm. The algorithm is n^2, and the tests are all done on the same n^2 algorithm without any algorithmic changes.

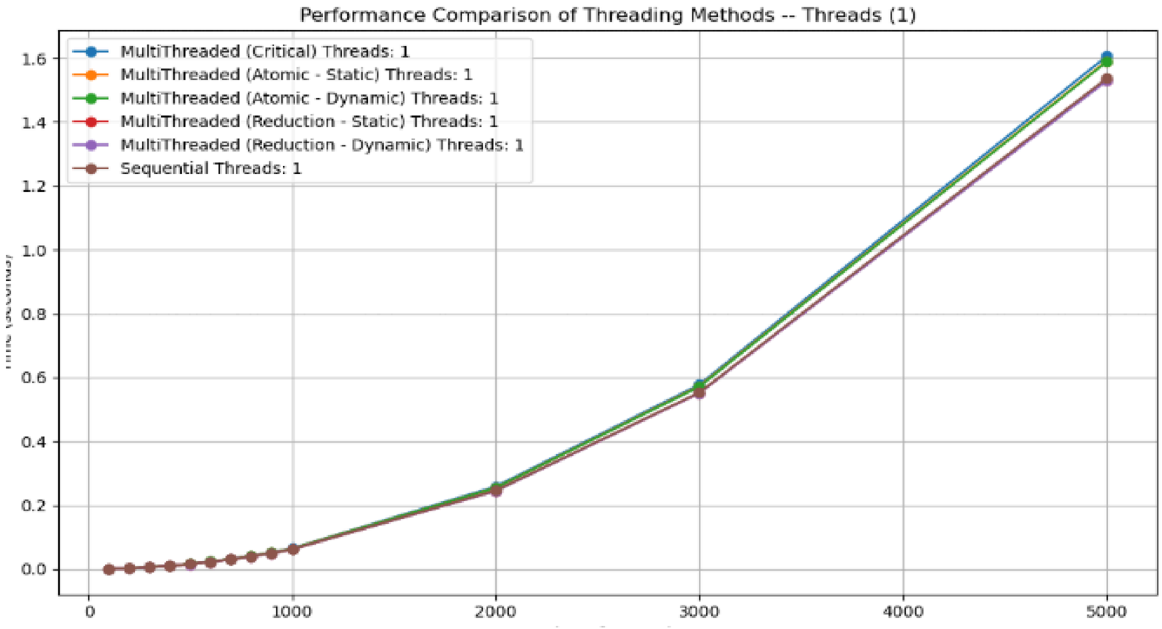
The first, and most obvious, fact that you can clearly observe when looking at how each method scales with respect to the number of threads, is that the number of threads does NOT equal better performance when multithreading.

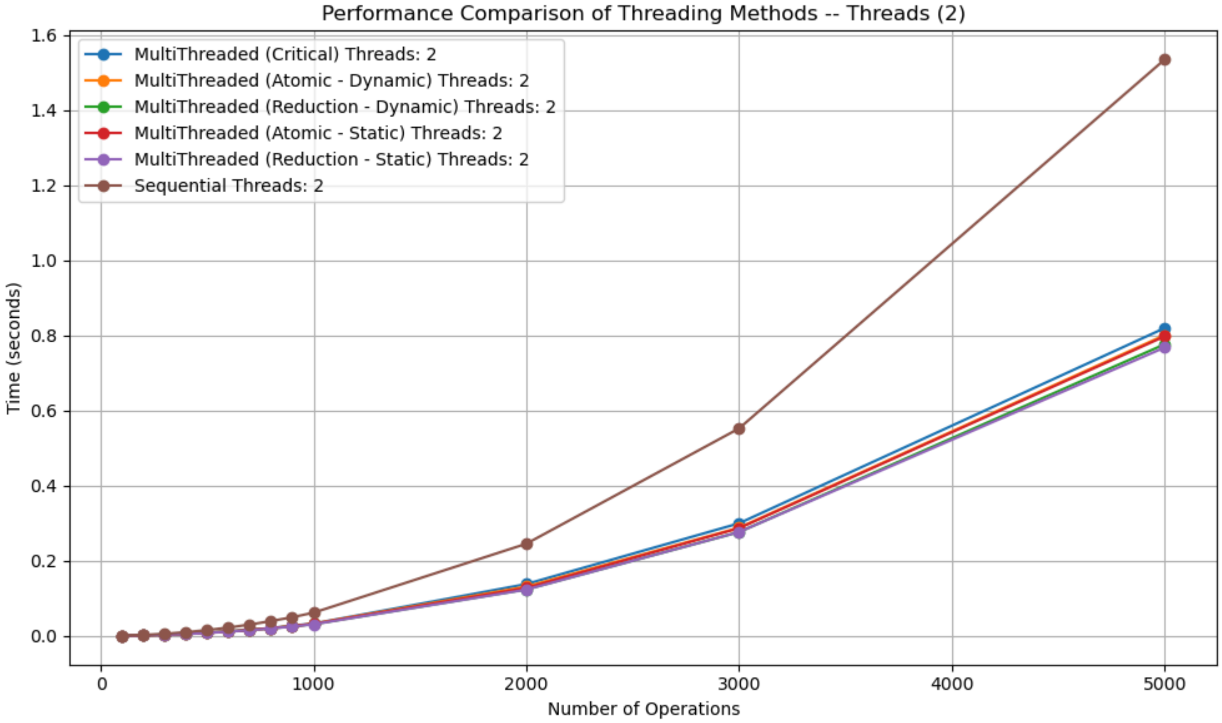
In fact, if you look at the Multithreaded Critical method, it actually scales way worse than sequential does. With respect to the number of threads and number of bodies. Overall, it is very bad for this particular use-case.

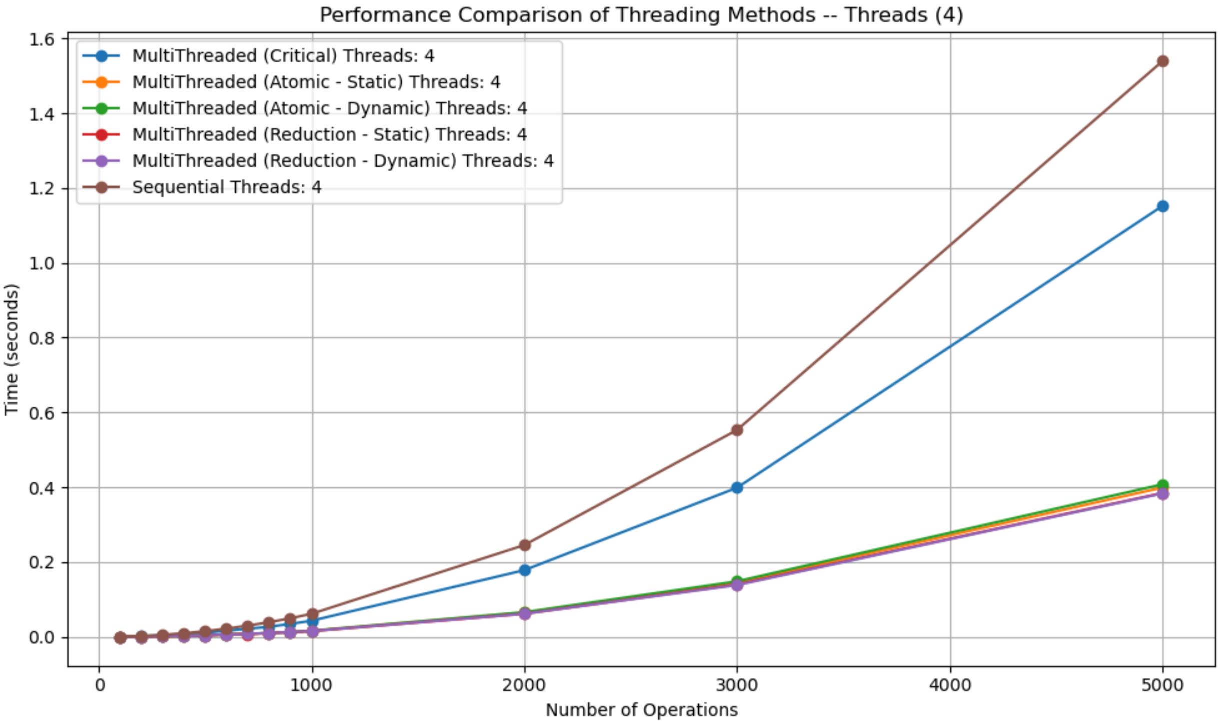
To learn the reason for this, we have to understand what the critical parameter does in omp. It is just a mutex. Mutexes are likely the first synchronization tool that you learn when learning about multithreading, and they are simple to use. However, they require that threads wait for access to specific portions of the algorithm and data. This means that while they are waiting, they cannot continue to do work.

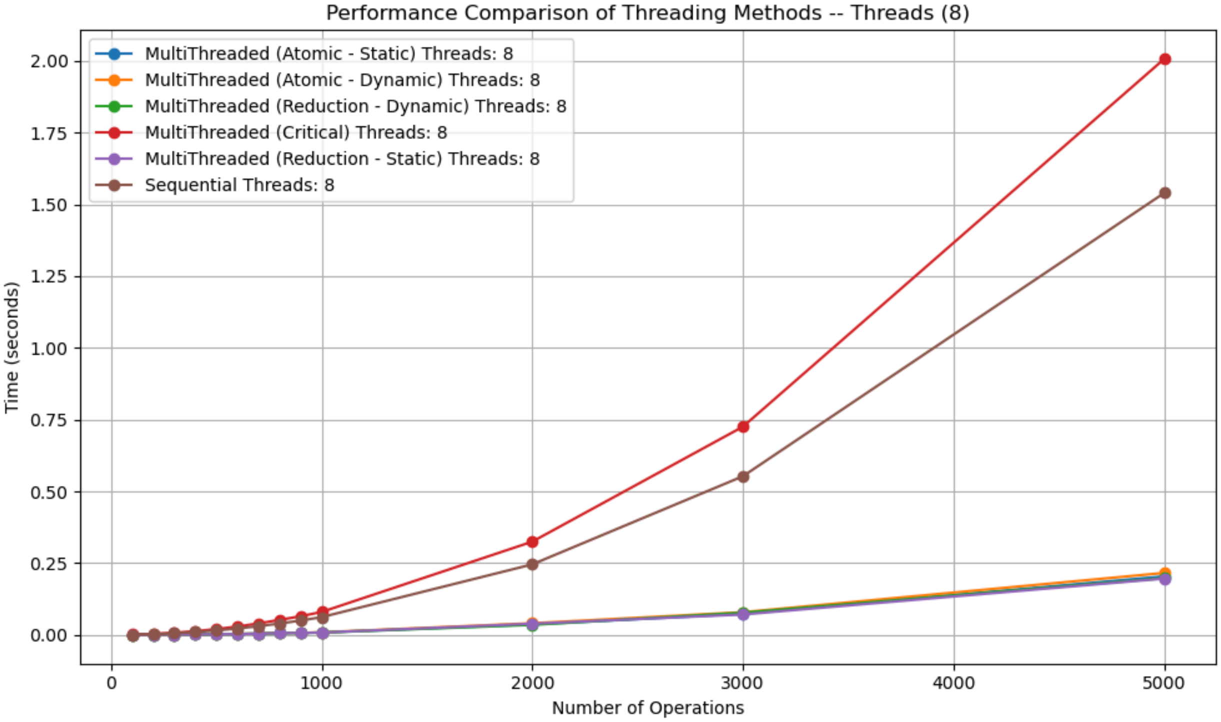
This makes sense as to why it scales poorly in terms of bodies, but why does it scale so badly in terms of many threads? One theory I have is that because there are more threads in total, there are more threads in line to access this special part of the algorithm. This means that the nth thread must wait for n-1 other threads to finish before it can continue. But why is this so much worse than sequential? My hypothesis is because these threads may be put to sleep by the operating system until its turn, or simply because this waiting requires the OS to continuously context switch and ruins all of the caching that the threads may or may not have done. There is some overhead when switching threads, so the sequential version has the advantage of not having to context switch as much and have better cache hits.

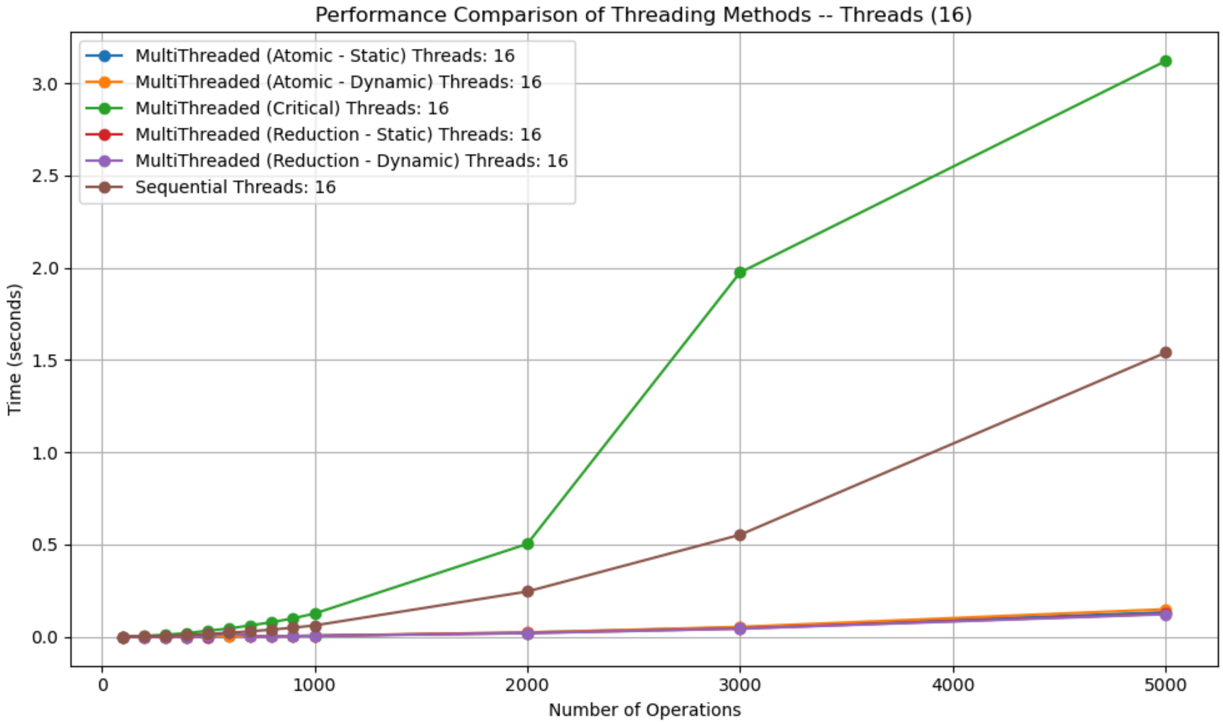
This is the most obvious interesting point from the data. The other MultiThreading methods (static vs dynamic, reduction vs atomic) all scale very similarly. Approaching a curve that looks close to a low coefficient linear curve, as opposed to an n^2 curve that the sequential has. Of course this makes sense as these methods allow all n threads to work at the same time, essentially approaching O(n) time as m threads approach n bodies.

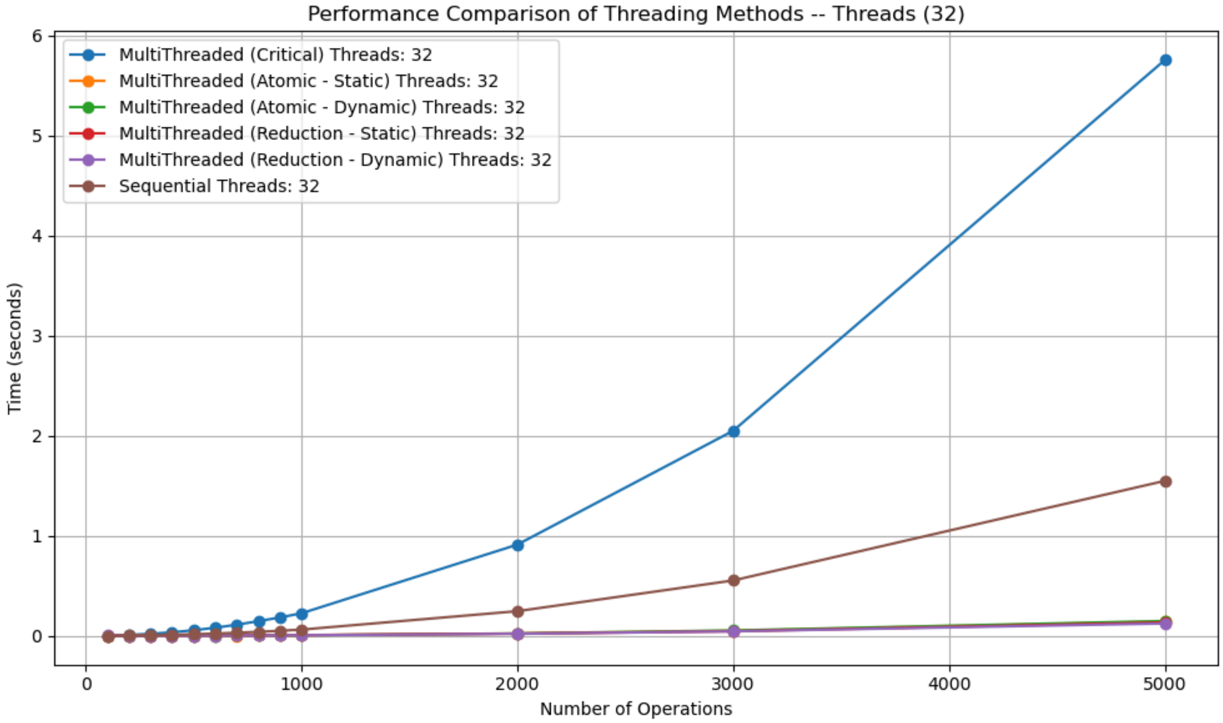

Performance Comparison of Threading Methods -- Threads (1)
MultiThreaded (Critical) Threads: 1
MultiThreaded (Atomic - Static) Threads: 1
MultiThreaded (Atomic - Dynamic) Threads: 1
MultiThreaded (Reduction - Static) Threads: 1
MultiThreaded (Reduction - Dynamic) Threads: 1
Sequential Threads: 1
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1000
2000
3000
4000
5000

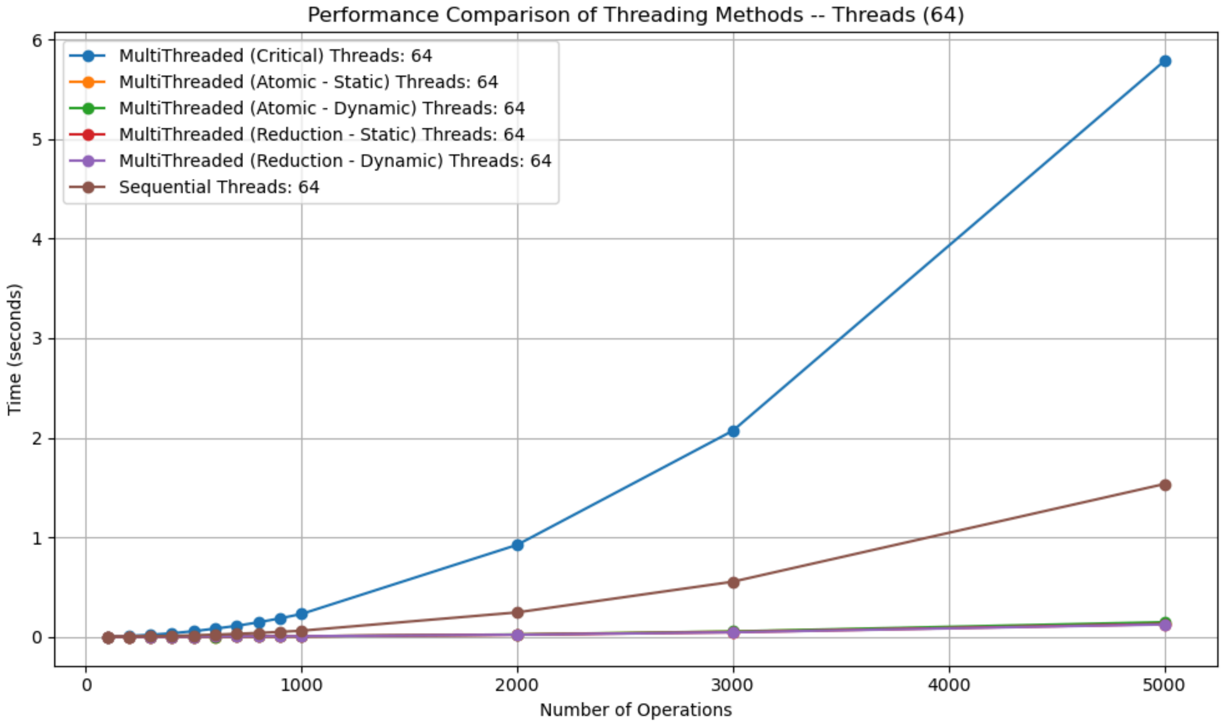

Performance Comparison of Threading Methods -- Threads (2)
MultiThreaded (Critical) Threads: 2
MultiThreaded (Atomic - Dynamic) Threads: 2
MultiThreaded (Reduction - Dynamic) Threads: 2
MultiThreaded (Atomic - Static) Threads: 2
MultiThreaded (Reduction - Static) Threads: 2
Sequential Threads: 2
Time (seconds)
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1000
2000
3000
4000
5000
Number of Operations

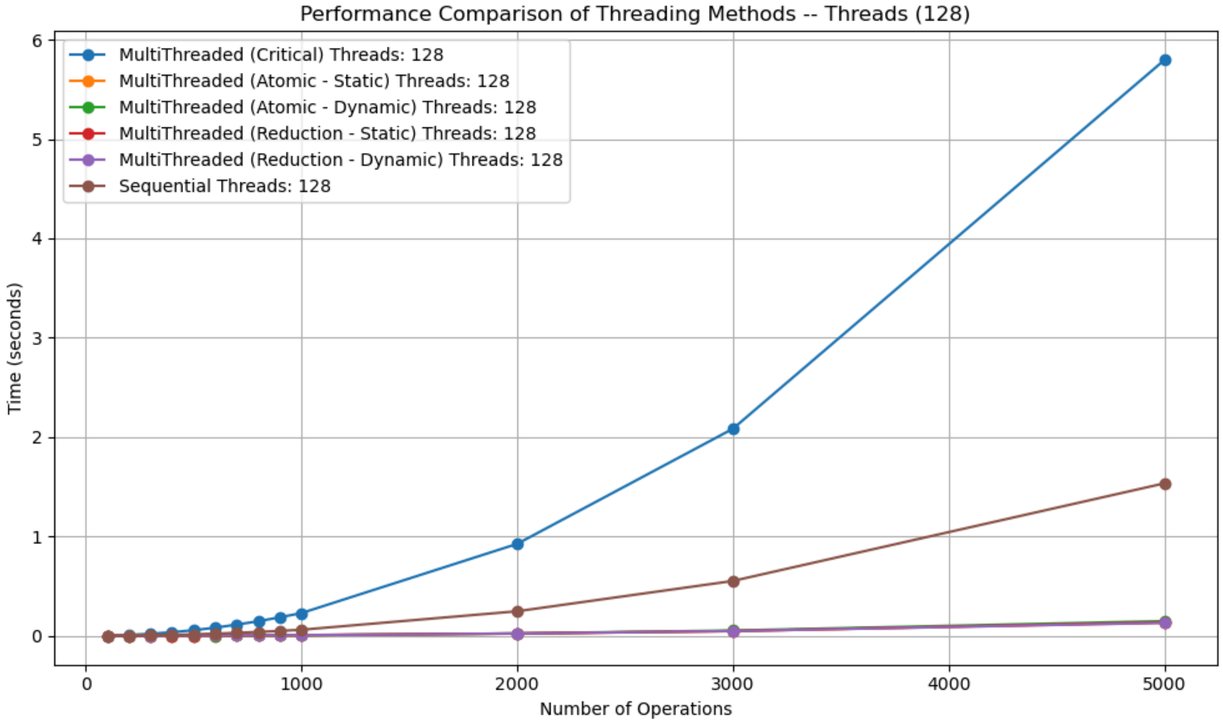

Performance Comparison of Threading Methods -- Threads (4)
MultiThreaded (Critical) Threads: 4
MultiThreaded (Atomic - Static) Threads: 4
MultiThreaded (Atomic - Dynamic) Threads: 4
MultiThreaded (Reduction - Static) Threads: 4
MultiThreaded (Reduction - Dynamic) Threads: 4
Sequential Threads: 4
Time (seconds)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
1000
2000
3000
4000
5000
Number of Operations


Performance Comparison of Threading Methods -- Threads (8)
MultiThreaded (Atomic - Static) Threads: 8
MultiThreaded (Atomic - Dynamic) Threads: 8
MultiThreaded (Reduction - Dynamic) Threads: 8
MultiThreaded (Critical) Threads: 8
MultiThreaded (Reduction - Static) Threads: 8
Sequential Threads: 8
Time (seconds)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
1000
2000
3000
4000
5000
Number of Operations


Performance Comparison of Threading Methods -- Threads (16)
MultiThreaded (Atomic - Static) Threads: 16
MultiThreaded (Atomic - Dynamic) Threads: 16
MultiThreaded (Critical) Threads: 16
MultiThreaded (Reduction - Static) Threads: 16
MultiThreaded (Reduction - Dynamic) Threads: 16
Sequential Threads: 16
Time (seconds)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
1000
2000
3000
4000
5000
Number of Operations


Performance Comparison of Threading Methods -- Threads (32)
MultiThreaded (Critical) Threads: 32
MultiThreaded (Atomic - Static) Threads: 32
MultiThreaded (Atomic - Dynamic) Threads: 32
MultiThreaded (Reduction - Static) Threads: 32
MultiThreaded (Reduction - Dynamic) Threads: 32
Sequential Threads: 32
Time (seconds)
0
1
2
3
4
5
6
0
1000
2000
3000
4000
5000
Number of Operations


Performance Comparison of Threading Methods -- Threads (64)
MultiThreaded (Critical) Threads: 64
MultiThreaded (Atomic - Static) Threads: 64
MultiThreaded (Atomic - Dynamic) Threads: 64
MultiThreaded (Reduction - Static) Threads: 64
MultiThreaded (Reduction - Dynamic) Threads: 64
Sequential Threads: 64
Time (seconds)
0
1
2
3
4
5
6
0
1000
2000
3000
4000
5000
Number of Operations


Performance Comparison of Threading Methods -- Threads (128)
MultiThreaded (Critical) Threads: 128
MultiThreaded (Atomic - Static) Threads: 128
MultiThreaded (Atomic - Dynamic) Threads: 128
MultiThreaded (Reduction - Static) Threads: 128
MultiThreaded (Reduction - Dynamic) Threads: 128
Sequential Threads: 128
Time (seconds)
0
1
2
3
4
5
6
0
1000
2000
3000
4000
5000
Number of Operations