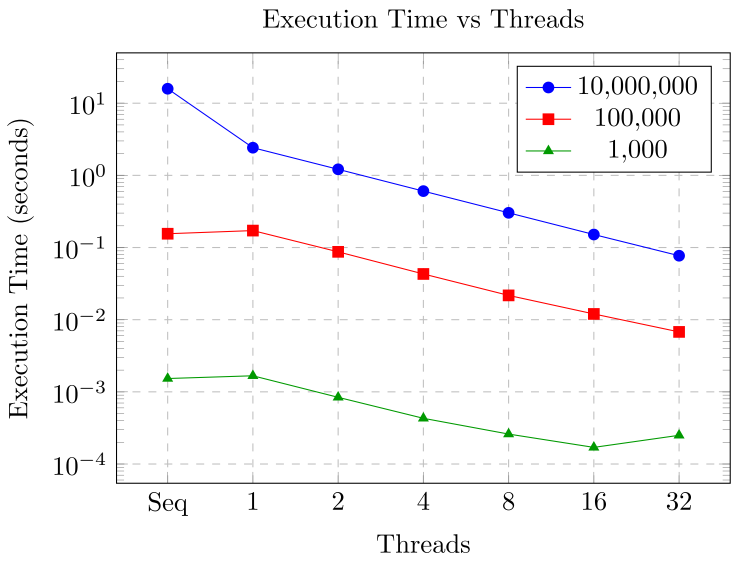
I couldn't get the Intel compiler to work, so the following results are only with gcc: Overall, performance was as expected, with an inverse linear relationship between threads and execution time. However, there were some notable outliers: 1000 vector size with 32 threads being slower than 16 threads, and 10000000 vector size sequentially being much slower than with 1 OMP thread. For the first, it is likely explained by the overhead from managing 32 threads being greater than the time saved by the increased parallelism for such a small problem. For the second, I'm less sure, but I'd guess that the performance boost comes from either compiler optimization or better cache use by the single-threaded OMP code.

Threads	**Vector Size: 10,000,000**	**Vector Size: 100,000**	**Vector Size: 1,000**
Sequential	15.86842	0.15527	0.00153
1	2.41733	0.17170	0.00167
2	1.21294	0.08716	0.00084
4	0.60504	0.04301	0.00043
8	0.30260	0.02171	0.00026
16	0.15144	0.01201	0.00017
32	0.07700	0.00677	0.00025


Execution Time vs Threads
Execution Time (seconds)
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
Seq
Threads