Harun Uğurlu 28532046358

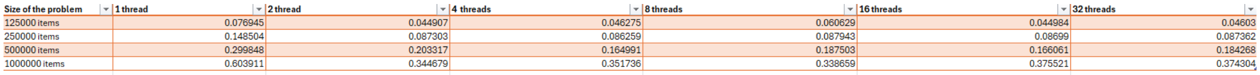
Ezgi Gülce Yazıcı 28925575182

Merge Sort Algorithm

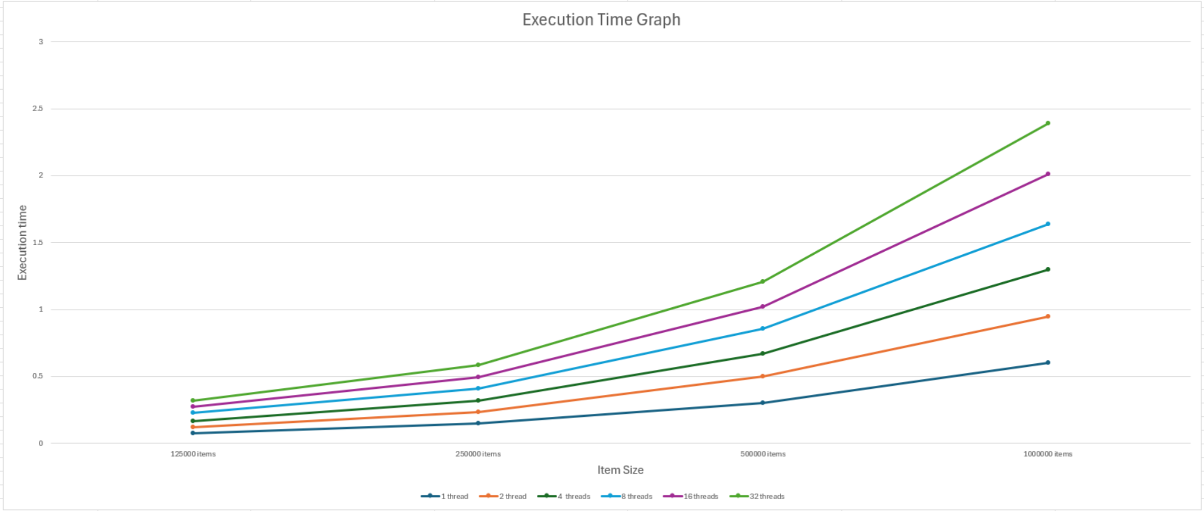
Introduction

In this assignment we have parallelized the merge sort algorithm using OpenMP library. Experimenting with different number of threads and different number of array sizes, we have analyzed the speed-up, efficiency of the algorithm and constructed tables and graphs to better visualize the results. Also, using these tables and graphs we have analyzed the scalability property of the algorithm.

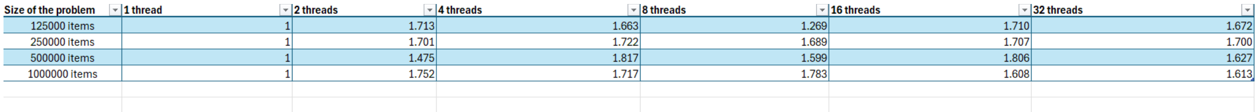
Execution Time:



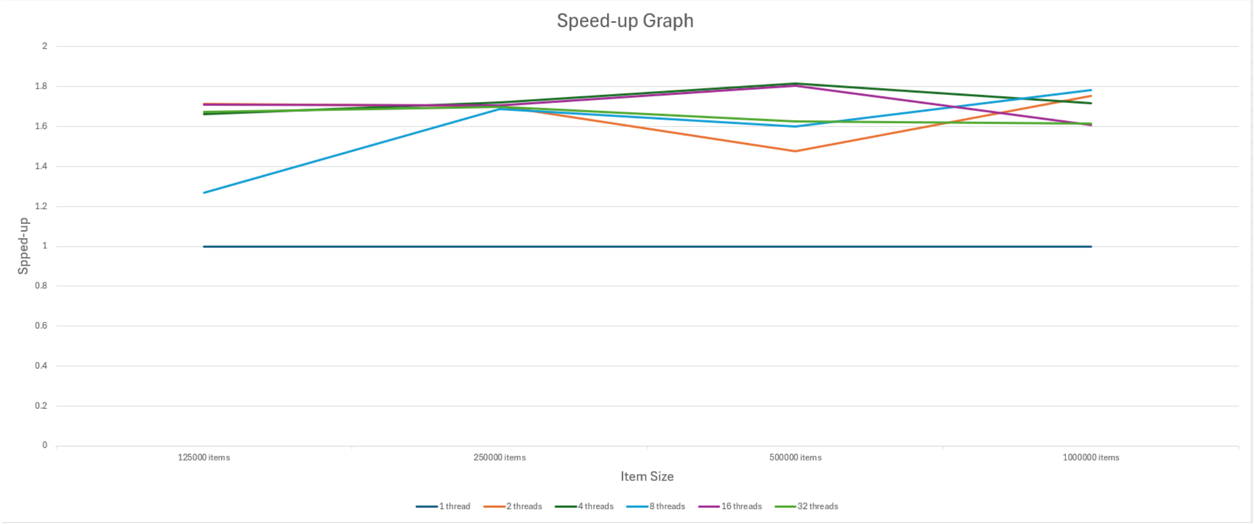
Size of the problem	1 thread	2 thread	4 threads	8 threads	16 threads	32 threads
125000 items	0.076945	0.044907	0.046275	0.060629	0.044984	0.04603
250000 items	0.148504	0.087303	0.086259	0.087943	0.08699	0.087362
500000 items	0.299848	0.203317	0.164991	0.187503	0.166061	0.184268
1000000 items	0.603911	0.344679	0.351736	0.338659	0.375521	0.374304



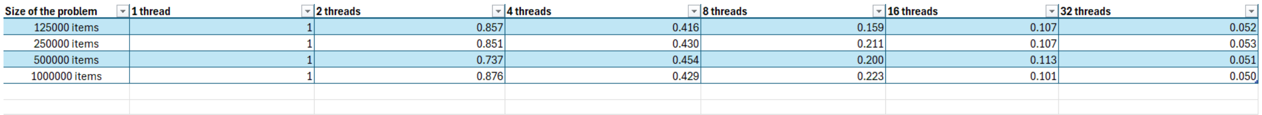
Speed-up:



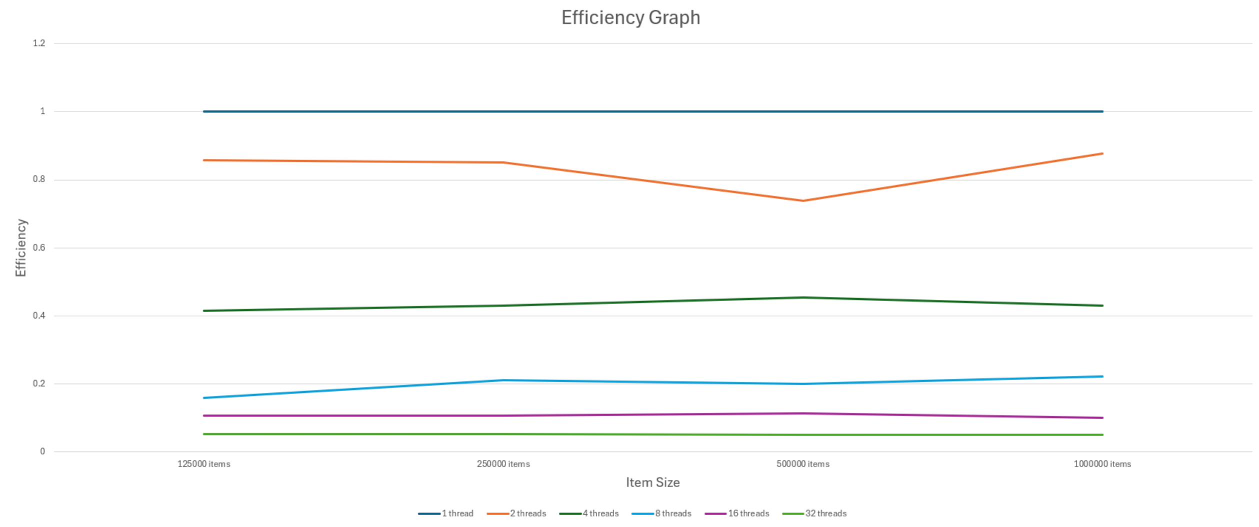
Size of the problem	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads
125000 items	1	1.713	1.663	1.269	1.710	1.672
250000 items	1	1.701	1.722	1.689	1.707	1.700
500000 items	1	1.475	1.817	1.599	1.806	1.627
1000000 items	1	1.752	1.717	1.783	1.608	1.613



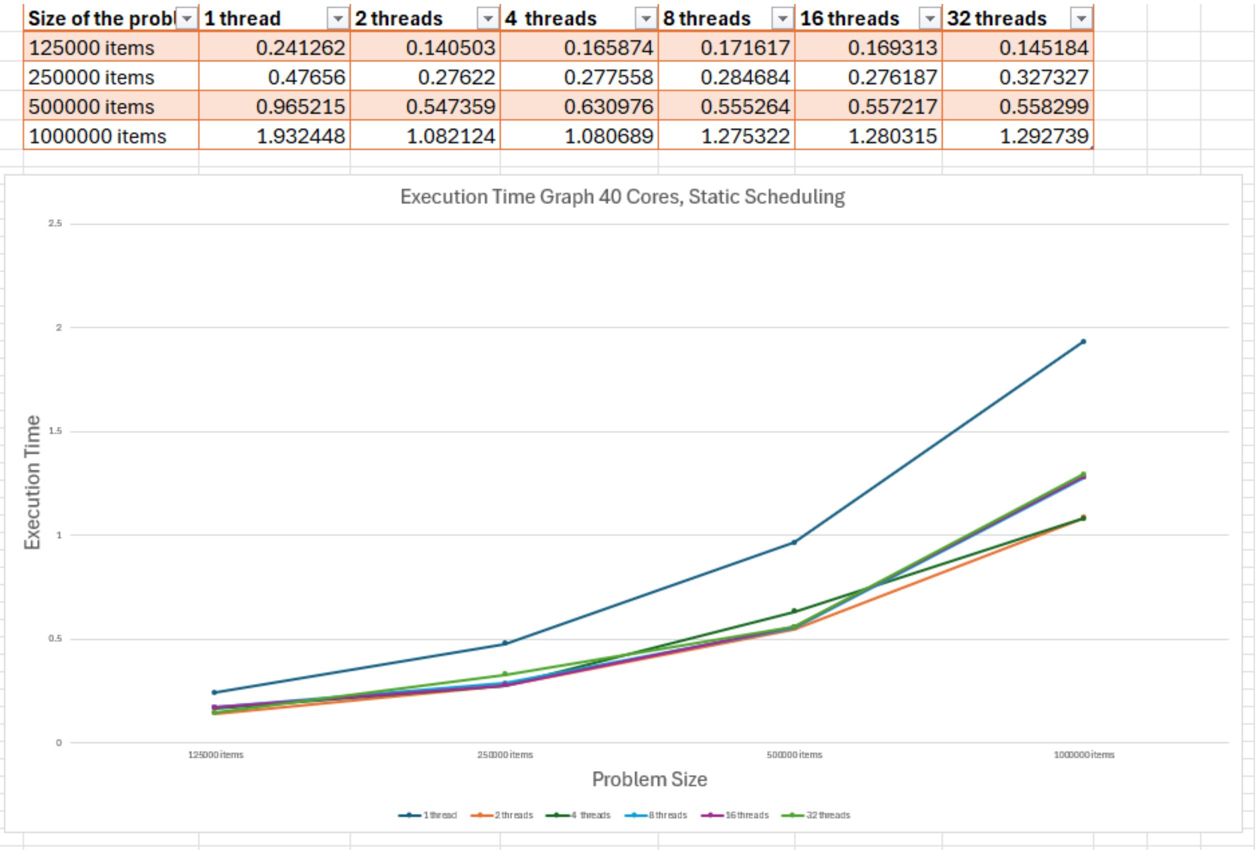
Efficiency:



Size of the problem	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads
125000 items	1	0.857	0.416	0.159	0.107	0.052
250000 items	1	0.851	0.430	0.211	0.107	0.053
500000 items	1	0.737	0.454	0.200	0.113	0.051
1000000 items	1	0.876	0.429	0.223	0.101	0.050

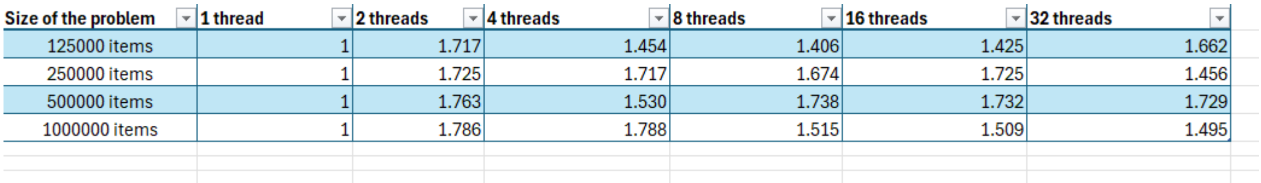


Execution Time 40 Cores, Static Scheduling:

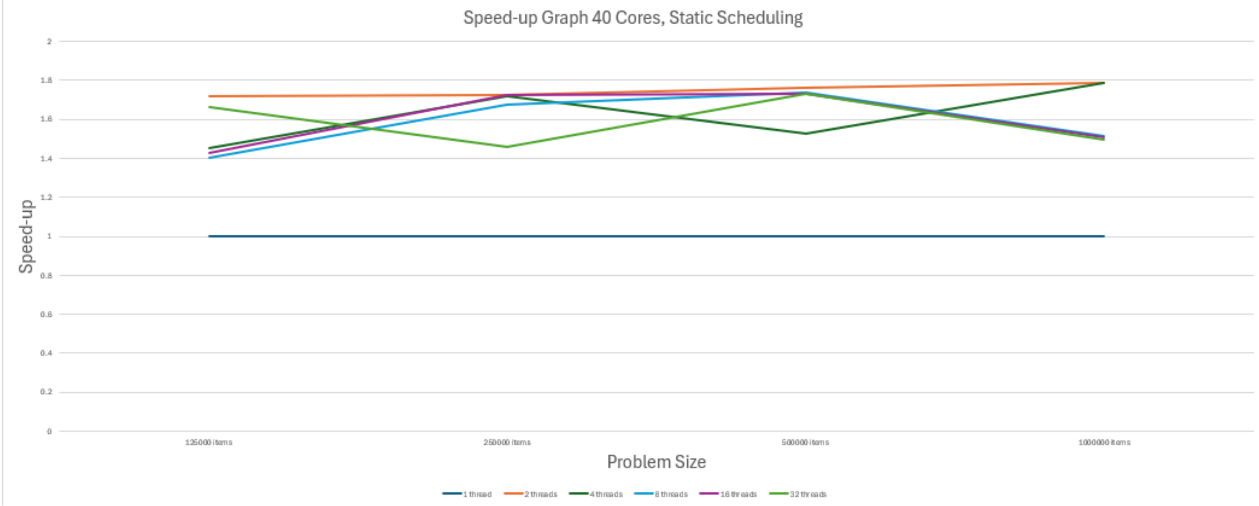


Size of the probl	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads
125000 items	0.241262	0.140503	0.165874	0.171617	0.169313	0.145184
250000 items	0.47656	0.27622	0.277558	0.284684	0.276187	0.327327
500000 items	0.965215	0.547359	0.630976	0.555264	0.557217	0.558299
1000000 items	1.932448	1.082124	1.080689	1.275322	1.280315	1.292739

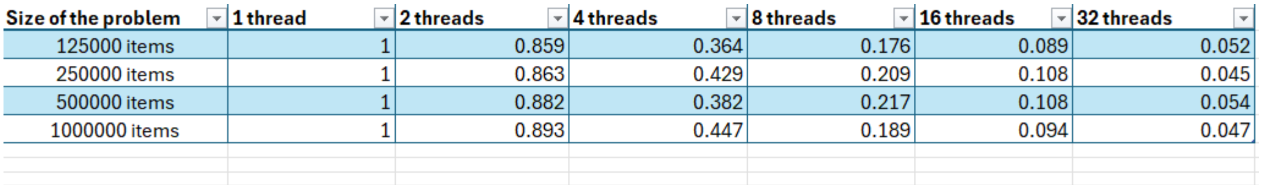
Speed-up Graph 40 Cores, Static Scheduling:



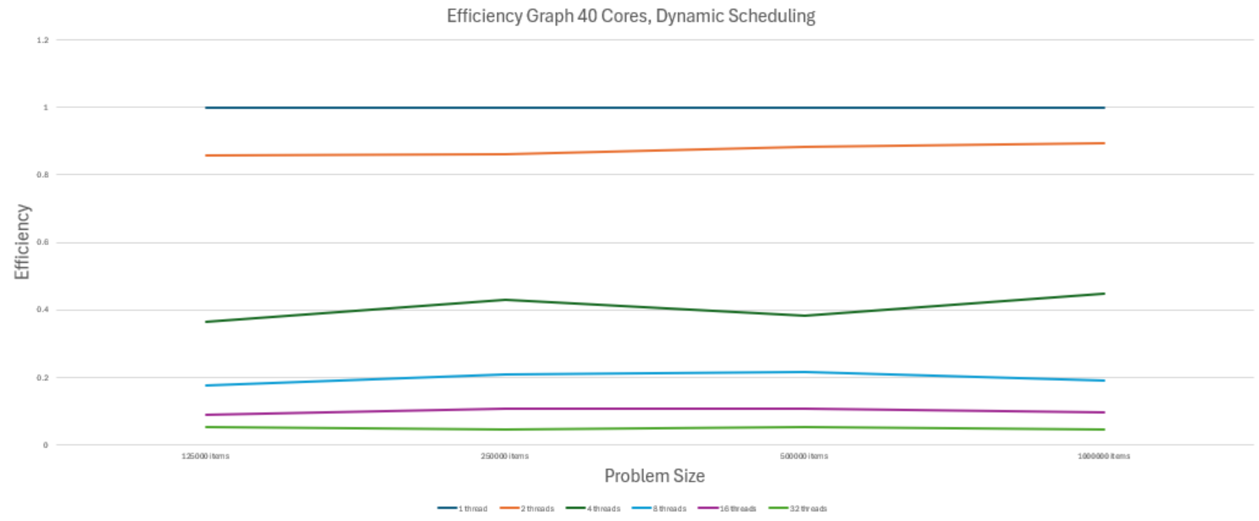
Size of the problem	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads
125000 items	1	1.717	1.454	1.406	1.425	1.662
250000 items	1	1.725	1.717	1.674	1.725	1.456
500000 items	1	1.763	1.530	1.738	1.732	1.729
1000000 items	1	1.786	1.788	1.515	1.509	1.495



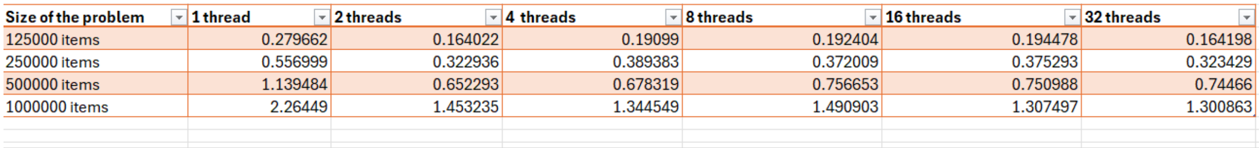
Efficiency Graph 40 Cores, Static Scheduling:



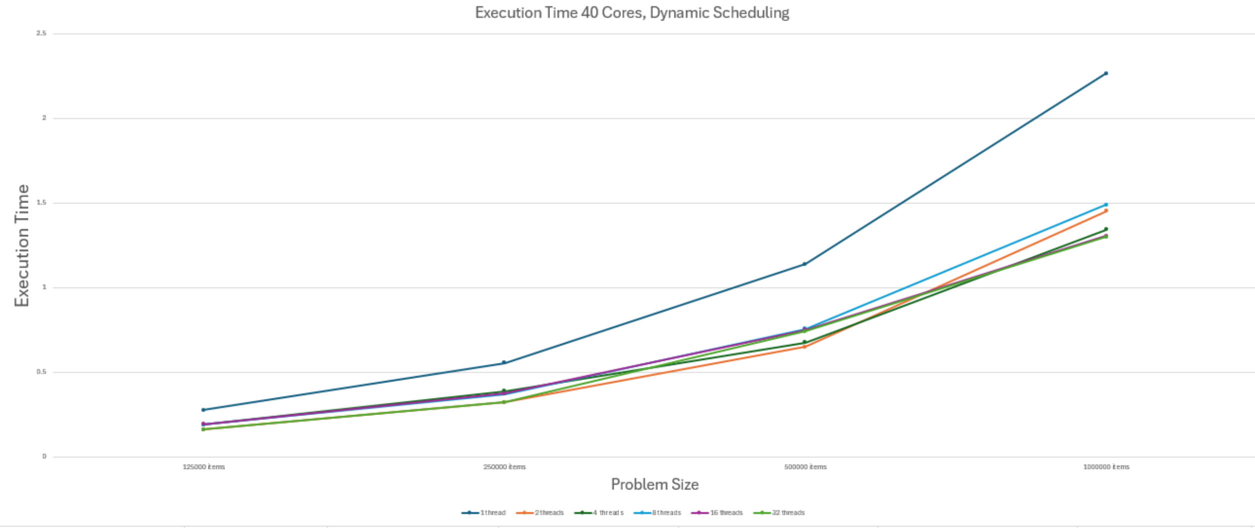
Size of the problem	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads
125000 items	1	0.859	0.364	0.176	0.089	0.052
250000 items	1	0.863	0.429	0.209	0.108	0.045
500000 items	1	0.882	0.382	0.217	0.108	0.054
1000000 items	1	0.893	0.447	0.189	0.094	0.047



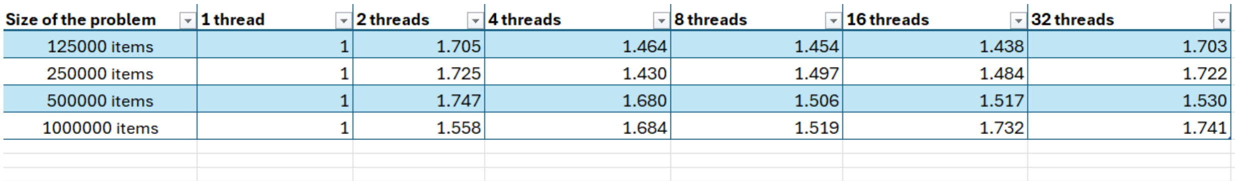
Execution Time 40 Cores, Dynamic Scheduling:



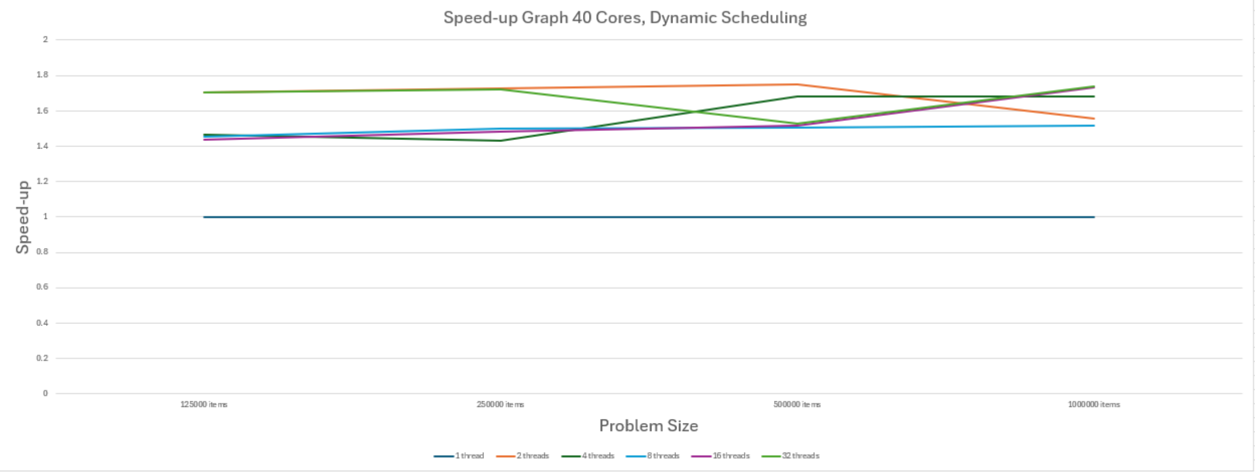
Size of the problem	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads
125000 items	0.279662	0.164022	0.19099	0.192404	0.194478	0.164198
250000 items	0.556999	0.322936	0.389383	0.372009	0.375293	0.323429
500000 items	1.139484	0.652293	0.678319	0.756653	0.750988	0.74466
1000000 items	2.26449	1.453235	1.344549	1.490903	1.307497	1.300863



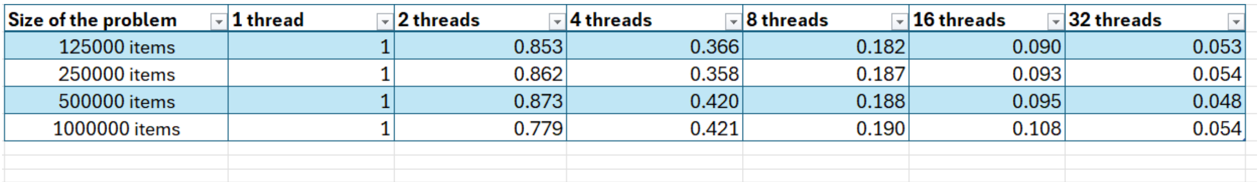
Speed-up Graph 40 Cores, Dynamic Scheduling:



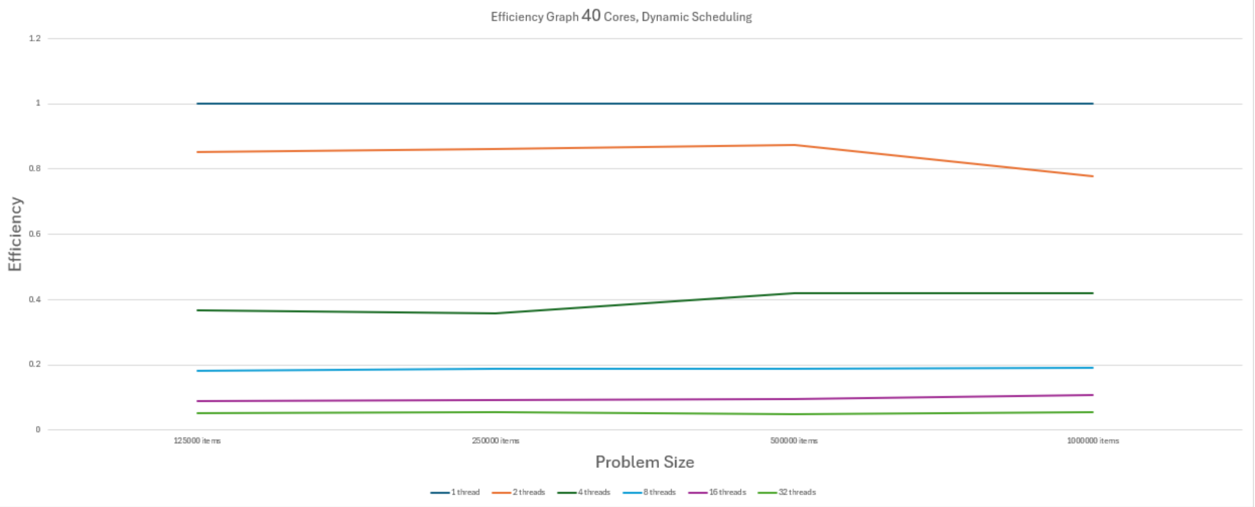
Size of the problem	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads
125000 items	1	1.705	1.464	1.454	1.438	1.703
250000 items	1	1.725	1.430	1.497	1.484	1.722
500000 items	1	1.747	1.680	1.506	1.517	1.530
1000000 items	1	1.558	1.684	1.519	1.732	1.741



Efficiency Graph 40 Cores, Dynamic Scheduling:



Size of the problem	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads
125000 items	1	0.853	0.366	0.182	0.090	0.053
250000 items	1	0.862	0.358	0.187	0.093	0.054
500000 items	1	0.873	0.420	0.188	0.095	0.048
1000000 items	1	0.779	0.421	0.190	0.108	0.054



Strong Scalability

If we increase the number of processes/threads and keep the efficiency fixed without increasing problem size, the problem is **strongly scalable**.

Based on the results, as we increase the number of threads, the efficiency doesn't stay fixed. Therefore, it is **not strongly scalable.**

Weak Scalability

If we keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes/threads, the problem is **weakly scalable.**

Based on the results, as we increase the problem size and the number of threads by the factor of 2, the efficiency doesn't stay fixed. Therefore, it is **not weakly scalable.**