Runtimes for different matrix sizes (All measurements in milliseconds)

MATRIX SIZE	**64** Parallel	**64** Non Parallel	**256** Parallel	**256** Non Parallel	**512** Parallel	**512** Non parallel	**1024** Parallel	**1024** Non Parallel	**4096** Parallel	**4096** Non Parallel
Test 1	1.1485	1.0999	3.9476	25.3707	30.1699	141.4117	342.0972	817.2175	19274.2597	43540.210
Test 2	0.9754	1.7119	2.9683	14.681	46.7299	119.3664	335.563	703.6633	18437.7669	42766.005
Test 3	0.8442	1.6782	2.7219	19.047	31.0579	113.3043	397.6555	686.0481	18391.5868	41543.654
Test 4	0.8283	1.7583	2.7232	17.5985	30.8901	133.621	314.1096	685.0236	19006.9746	42873.765
Test 5	0.7835	0.9111	2.6707	17.6134	39.5226	139.9937	253.9623	684.8679	18006.1815	40123.643
Test 6	0.8288	1.164	2.1844	15.1331	44.3544	115.2225	246.8204	684.3465	18867.0973	42956.342
Test 7	0.8106	1.3889	2.8243	50.7674	35.987	112.6726	246.3638	680.904	18668.5884	44542.432
Test 8	0.8386	1.3904	4.8751	11.0645	37.6261	138.762	278.892	699.666	18435.3245	44421.765
Test 9	0.8821	0.805	2.5435	12.8062	36.6097	188.3386	251.009	722.1668	19123.9135	40543.657
Test 10	0.903	0.8914	2.5239	12.0376	35.2955	110.1061	245.7517	679.3839	17645.3241	43432.876
AVERAGE	0.8843	1.2799	2.9982	19.6119	36.8243	131.279	291.222	704.328	18585.7	42674.4

The runtimes compared on a logarithmic scale favor the parallel algorithm. The biggest percentage increase of performance happened at matrix size 256. This is likely due to the serial part of the program getting more intense and balancing out as the size increased more. However the parallel program vastly outperforms for each matrix size.