Each is done with 100 generations.
Displays at generation 0, and then every 10 generations.

Total Time

	Input Sizes (nxn					
# of Processes	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	0.131	0.704	4.679	33.737999	257.041992	N/A	N/A
2	0.061	0.293	1.601	10.667	78.282997	N/A	N/A
4	0.04	0.167	0.83	4.559	29.194	N/A	N/A
8	N/A	0.813	1.155	3.232	15.07	96.13	N/A
16	N/A	N/A	3.685	5.259	12.68	59.4	377.601

Average Display Time

	Input Sizes (nxn						
# of Processes	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	0.0003	0.0007	0.0026	0.0079	0.0272	N/A	N/A
2	0.0003	0.0006	0.0019	0.0007	0.0233	N/A	N/A
4	0.0003	0.0007	0.0027	0.0114	0.0485	N/A	N/A
8	N/A	0.0007	0.0021	0.0158	0.0479	0.1135	N/A
16	N/A	N/A	0.0244	0.0389	0.0715	0.1332	0.8445

Total Time - Display Time

	Input Sizes (nxn						
# of Processes	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	0.128	0.697	4.653	33.658999	256.769992	N/A	N/A
2	0.058	0.287	1.582	10.66	78.049997	N/A	N/A
4	0.037	0.16	0.803	4.445	28.709	N/A	N/A
8	N/A	0.806	1.134	3.074	14.591	94.995	N/A
16	N/A	N/A	3.441	4.87	11.965	58.068	369.156

Average Time Per Generation

	Input Sizes (nxn						
# of Processes	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	0.00128	0.00697	0.04653	0.33658999	2.56769992	N/A	N/A
2	0.00058	0.00287	0.01582	0.1066	0.78049997	N/A	N/A
4	0.00037	0.0016	0.00803	0.04445	0.28709	N/A	N/A
8	N/A	0.00806	0.01134	0.03074	0.14591	0.94995	N/A
16	N/A	N/A	0.03441	0.0487	0.11965	0.58068	3.69156

Total Communication

	Input Sizes (nxn						
# of Processes	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	0	0	0	0	0	N/A	N/A
2	0.001	0.003	0.004	0.008	0.022	N/A	N/A
4	0.003	0.004	0.004	0.028	0.07	N/A	N/A
8	N/A	0.267	0.416	0.481	0.795	0.94	N/A
16	N/A	N/A	1.257	1.627	3.864	4.556	9.037

Total Computation

	Input Sizes (nxn						
# of Processes	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	0.128	0.697	4.653	33.658999	256.769992	N/A	N/A
2	0.057	0.284	1.578	10.652	78.027997	N/A	N/A
4	0.034	0.156	0.799	4.417	28.639	N/A	N/A
8	N/A	0.539	0.718	2.593	13.796	94.055	N/A
16	N/A	N/A	2.184	3.243	8.101	53.512	360.119

Speedup

	Input Sizes (nxn						
# of Processes	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	1	1	1	1	1	N/A	N/A
2	0.453125	0.4117647059	0.3399957017	0.3167057939	0.3039685299	N/A	N/A
4	0.2890625	0.2295552367	0.1725768322	0.1320597799	0.1118082365	N/A	N/A
8	N/A	1.156384505	0.2437137331	0.09132773081	0.05682517605	N/A	N/A
16	N/A	N/A	0.7395228885	0.144686418	0.04659812429	N/A	N/A

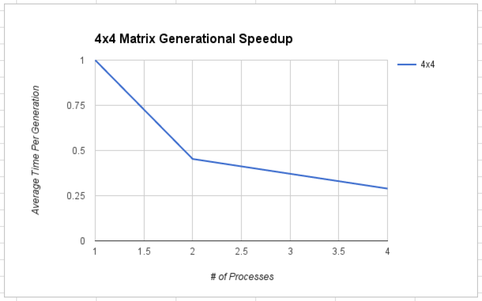
% of Time Computing

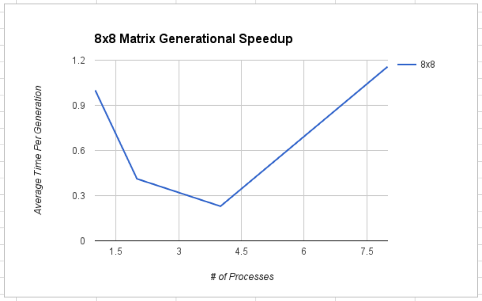
	Input Sizes (nxn						
# of Processes	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	100	100	100	100	100	N/A	N/A
2	98.27586207	98.95470383	99.7471555	99.9249531	99.97181294	N/A	N/A
4	91.89189189	97.5	99.501868	99.37007874	99.75617402	N/A	N/A
8	N/A	66.87344913	63.31569665	84.352635	94.55143582	N/A	N/A
16	N/A	N/A	63.46992153	66.59137577	67.70580861	N/A	N/A

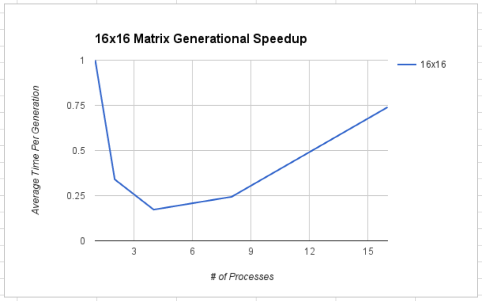
% of Time Communicating

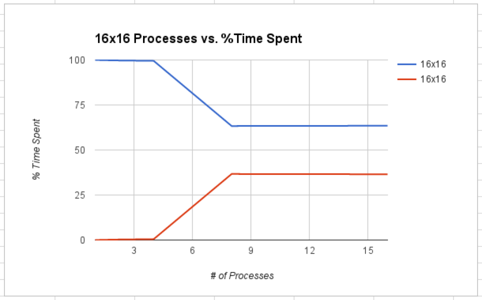
	Input Sizes (nxn

NOTE: I was unable to get this working on the GLX Nodes. As
an alternative I used Microsoft MPI in Visual Studio. Because of this
I was unable to use values of n larger than 256 due to the time it would
take for low numbers of processes. However the speedup graphs and
% time taken graphs are still legitimate due to them being fractional
values, and not reliant on total time taken.

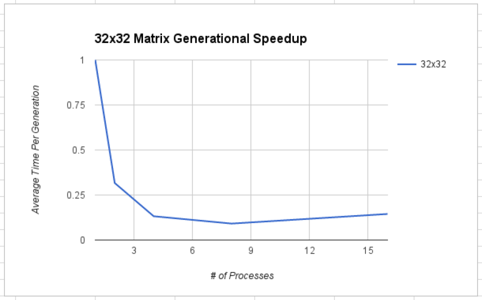


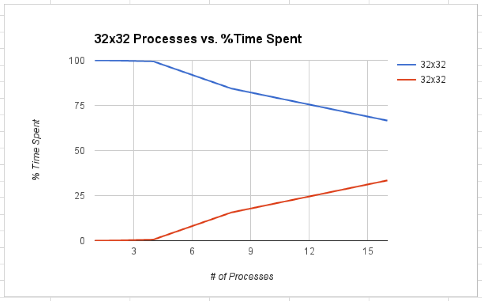


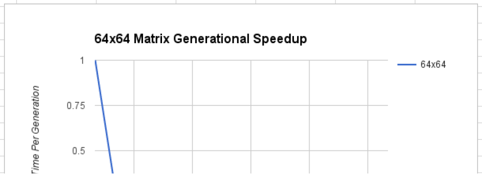


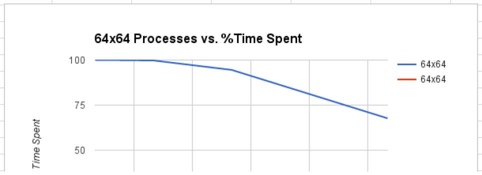


For the % Time S
is the time spent
is the time spent

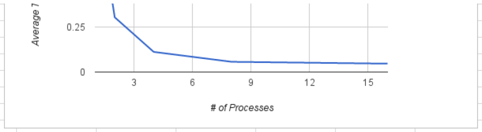


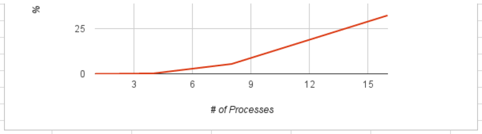






# of Processes	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	0	0	0	0	0	N/A	N/A
2	1.724137931	1.045296167	0.2528445006	0.07504690432	0.02818706066	N/A	N/A
4	8.108108108	2.5	0.498132005	0.6299212598	0.2438259779	N/A	N/A
8	N/A	33.12655087	36.68430335	15.647365	5.448564183	N/A	N/A
16	N/A	N/A	36.53007847	33.40862423	32.29419139	N/A	N/A





Analysis: These results meet my analytical expectations. In cases where n>>p twice the speedup of the previous # of processes was acheived. In all cases except the 64x64 matrix using 16 processes did not improve speedup because the time spent communicating was greater than the benefit of the reduced computational time. Although I did not run many instances of the 128x128 and 256x256 matrices, due to the massiv amount of time that they would require for low number of processes I predict that this is where using 16 processes would be vastly superior to lower numbers and we would start to see double speedup once more. When using 16 processes the % time spent communicating hovered around 33%, and as n increases I predict that this number would rise slightly, but level out at some point.