Michael Laucella
Lab1

3.)

Iterations:

Size	Gsref	Gs
2	7	7
4	9	9
8	12	12
16	14	14
32	1	17
64	18	18
128	18	18
256	20	20
512	20	20
1024	21	21
2048	23	23
4096	31	28

4.)

Timings

Processes

	1	2	4	8	16	32	64
2	0.665	0.6738					
4	0.6646	0.6724	0.6844				
8	0.6652	0.6752	0.6848	0.7138			
16	0.6668	0.676	0.7146	0.712	0.9382		

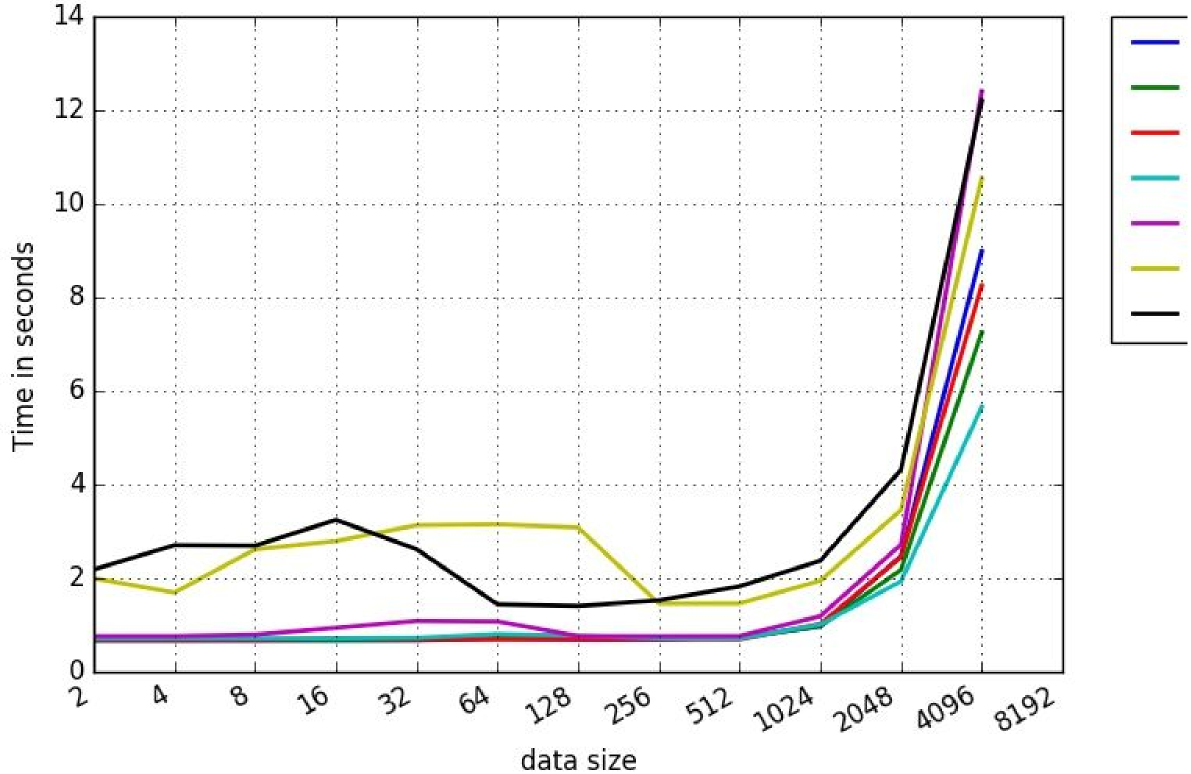
32	0.668	0.6786	0.6942	0.7272	1.0888	3.132	
64	0.7058	0.7154	0.6884	0.8058	1.0774	3.1506	1.442
128	0.6872	0.7136	0.691	0.7734	0.7634	3.085	1.4058
256	0.6884	0.7356	0.689	0.7132	0.752	1.459	1.5276
512	0.7386	0.7196	0.69	0.7098	0.7644	1.4612	1.8252
1024	0.9674	0.9756	1.0218	1.0112	1.1882	1.945	2.3726
2048	2.4734	2.1786	2.4482	1.9338	2.7186	3.4562	4.3152
4096	8.9824	7.2504	8.25	5.6576	12.404	10.5348	12.2034

5.)

Speedup

Processes

	1	2	4	8	16	32	64
2	1	.9869					
4	1	.9884	.9711				
8	1	.9852	.9714	.9319			
16	1	.9864	.9331	.9365	.7107		
32	1	.9844	.9623	.9186	.6135	.2133	
64	1	.9866	1.025	.8759	.6551	.2240	.4895
128	1	.9630	.9945	.8885	.9002	.2228	.4888
256	1	.9358	.9991	.9699	.9154	.4718	.4506
512	1	1.026	1.070	1.040	.9662	.5055	.4047
1024	1	.9916	.9468	.9567	.8142	.4974	.4077
2048	1	1.135	1.010	1.2790	.9098	.7156	.5732
4096	1	1.239	1.089	1.588	.7242	.8526	.7361



Note* the processes are ordered 1->64 from top to bottom of the legend on the right ex: blue(1)

6.)
Crunchy3
Threads per core: 1
Cores per socket: 8
Sockets: 4
Total: 4*1*8=32

7.)
Note that 16 processes with 4096 is the slowest, this was averaged over five times in a script however re-running placed it at around 5 making it fit in with the other curves.

a.) view the table above the red sections are no speedup sections:

All of 64,32,16
2 from data sizes: 2-256 and 1024
4 from data sizes: 4-32, 128-256 and 1024
8 from data sizes: 8-256 and 1024

b.) Knowing that communication has a greater effect on speed then computation, when there is lower amounts of data and a higher number of processes, the overhead slows down the system to such an extent that the serialized version is faster this is obvious. Now as there is an increase in the size of the data there should be an increase in performance in the lower spectrum moving to the higher number of processes as the data size increases. If the data sizes were to keep growing we should start to see 16 then 32 then 64 speeding up.

c.) view the table above the green sections represent speedup
2 from data sizes: 512, 2048, 4096
4 from data sizes: 64, 512, 2048, 4096
8 from data sizes: 512, 2048, 4096

d.) The speedup is following a consistent pattern, the lower number of processes are speeding up with higher datasets, this is likely due to a proper balance between the work done and the communication overheads. The overheads caused by communication require large amounts of time a Allreduce test on crunchy3 took approx .0001(s) on 2 processes and .003(s) on 64. From this it’s also clear that communication takes much longer when there are more processes that need to communicate as such the key to speedup here is that the data computation must take longer than the communication time, and as the data grows we get each process running O(loc_n*num*iterations) for the serial version it is running O(num^2*iterations) the num^2 will quickly overcome the loc_n*num and become more expensive than the communications even given more processes.