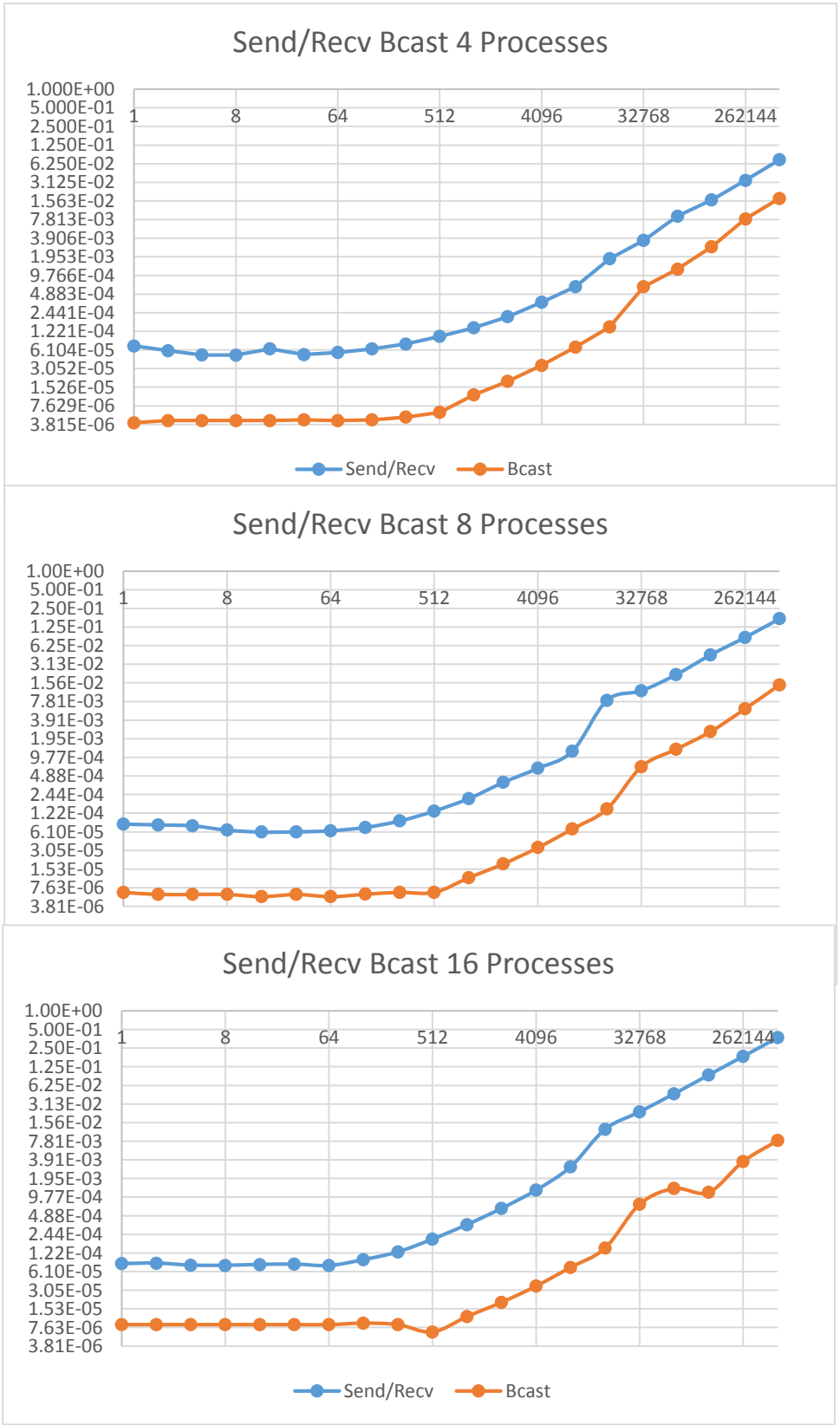
CS4234 P1
elobeto
Evan Lobeto

1.

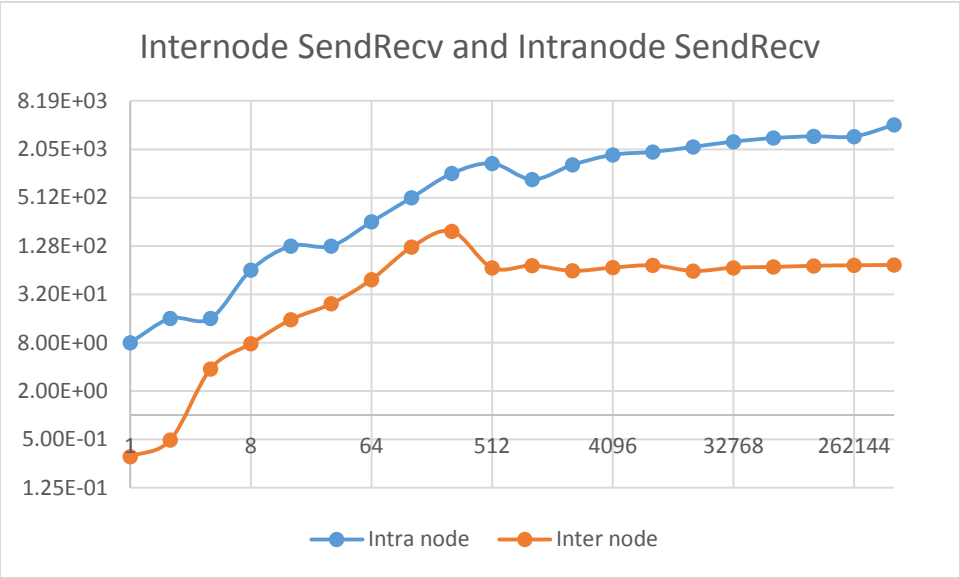
a.



CS4234 P1
elobeto
Evan Lobeto

From the charts shown above, it can be clearly seen that MPI_Bcast takes consistently less time to send messages to multiple processes than MPI_Send and MPI_Recv do on their own. This is most likely because MPI_Send and MPI_Recv have to be called multiple times to send data to each process whereas MPI_Bcast only needs to be called once.

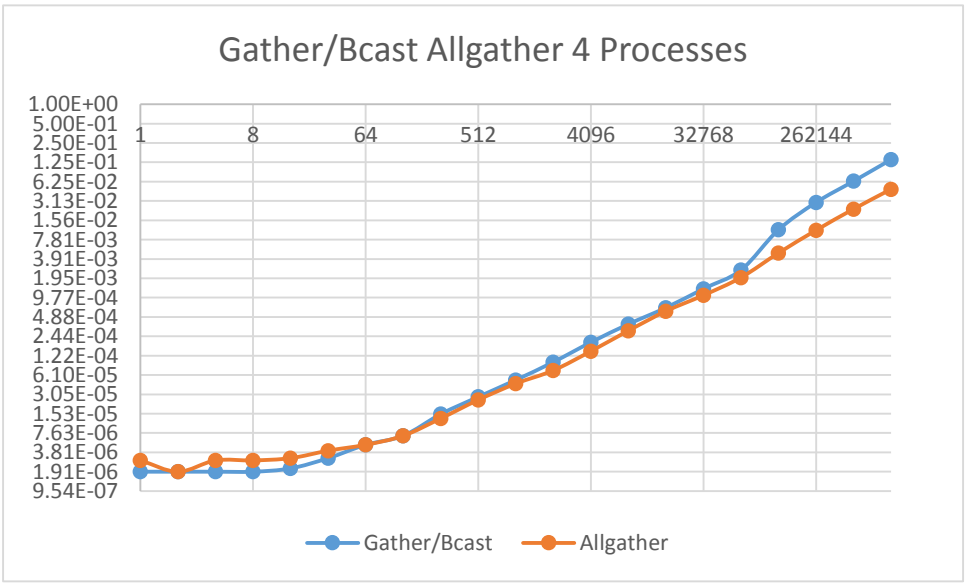
b.



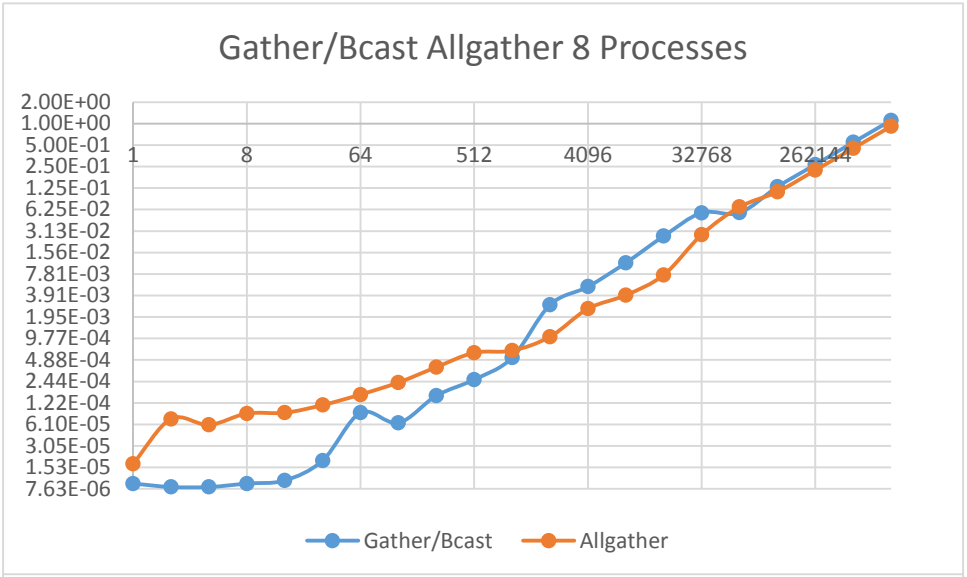
This chart shows that 2 MPI_Send calls from 2 processes to 2 other processes with MPI_Recv calls takes much more time from node to node than it does from core to core. This is expected because of the fact that the data needs to be sent over the network to travel between nodes where in one node it can just be retrieved from memory.

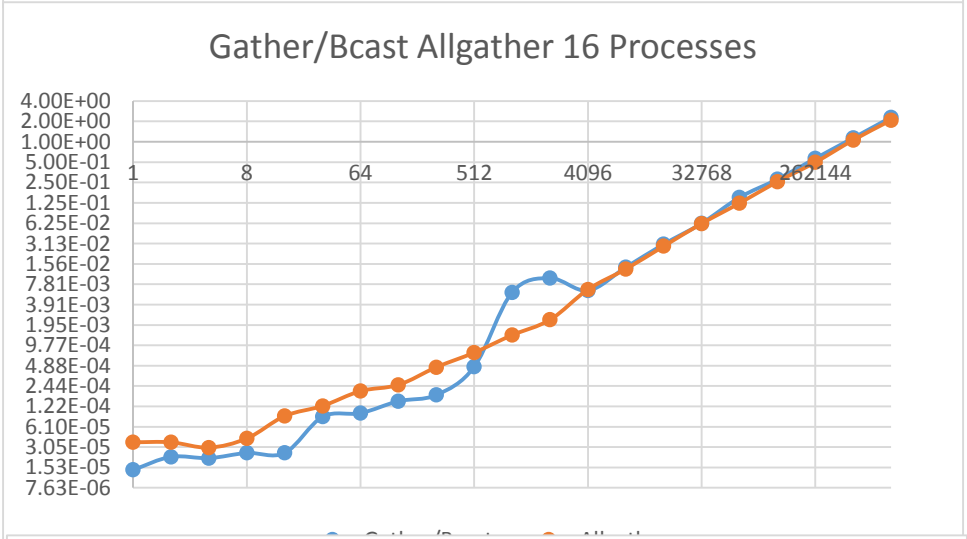
2.

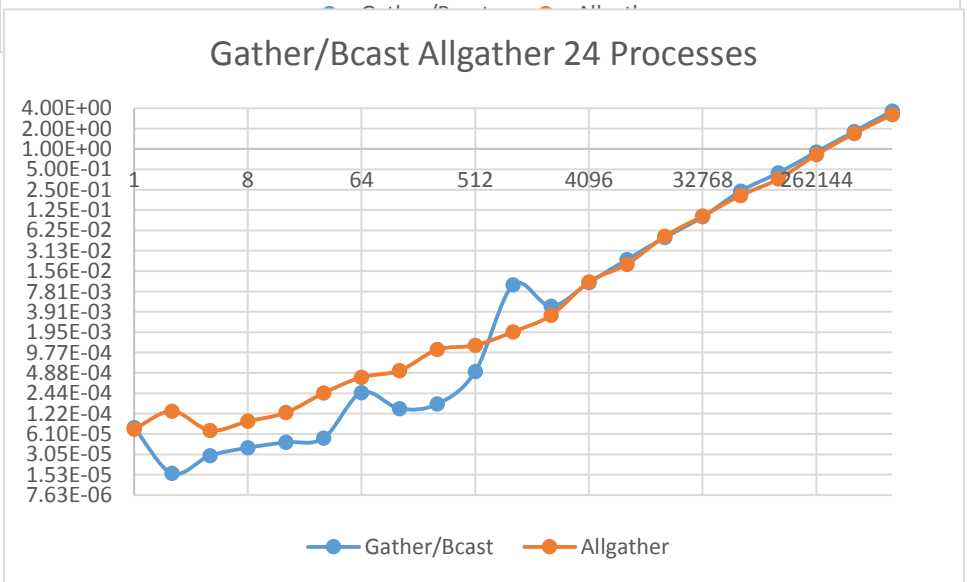
a.



CS4234 P1
elobeto
Evan Lobeto



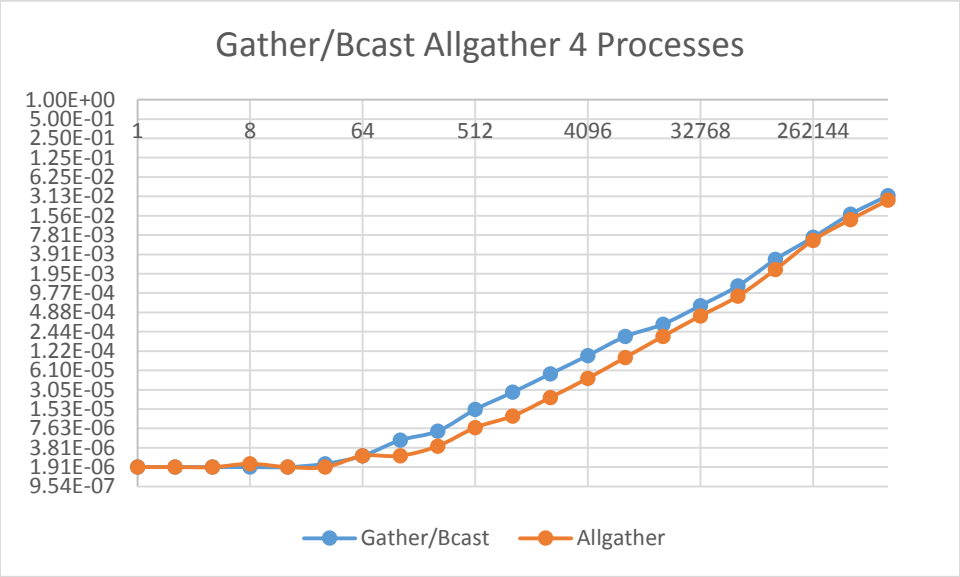


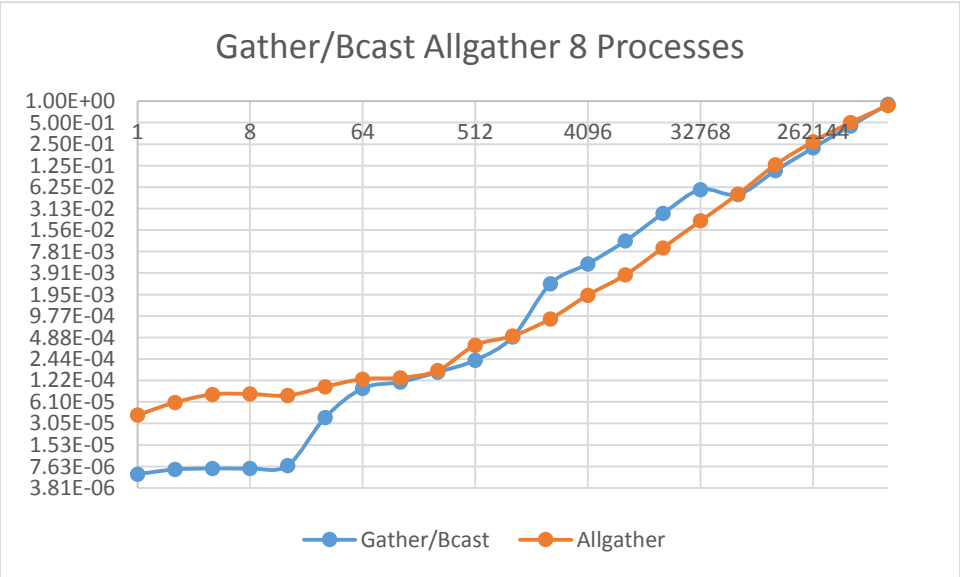


CS4234 P1
elobeto
Evan Lobeto

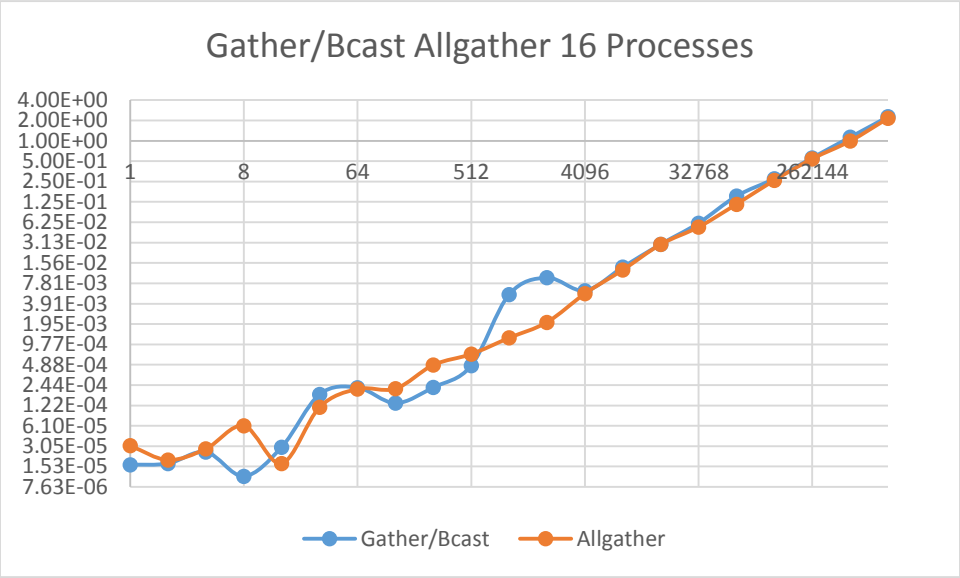
From the charts in 2a, it can be seen that using MPI_Gather to coalesce the data onto the master core and then calling MPI_Bcast to disperse it to all cores typically runs slightly faster than just using MPI_Allgather up until a message size of 1024 where they have similar performance. The MPI_Allgather call usually has a more consistent timing increase as message size increases than does MPI_Gather/MPI_Bcast but the variability of the rlogin cluster could have something to do with that since there are 2 calls to be made rather than just one for MPI_Allgather. Through the collection of charts, it can be seen that as the number of processes increases, so does the time. This is probably because there is more network traffic and more data to coalesce. They most likely have similar timings because they are essentially doing the same task and MPI_Gather/MPI_Bcast are global calls like MPI_Allgather in that they don't need to be called more than once to get/send data from all processors.

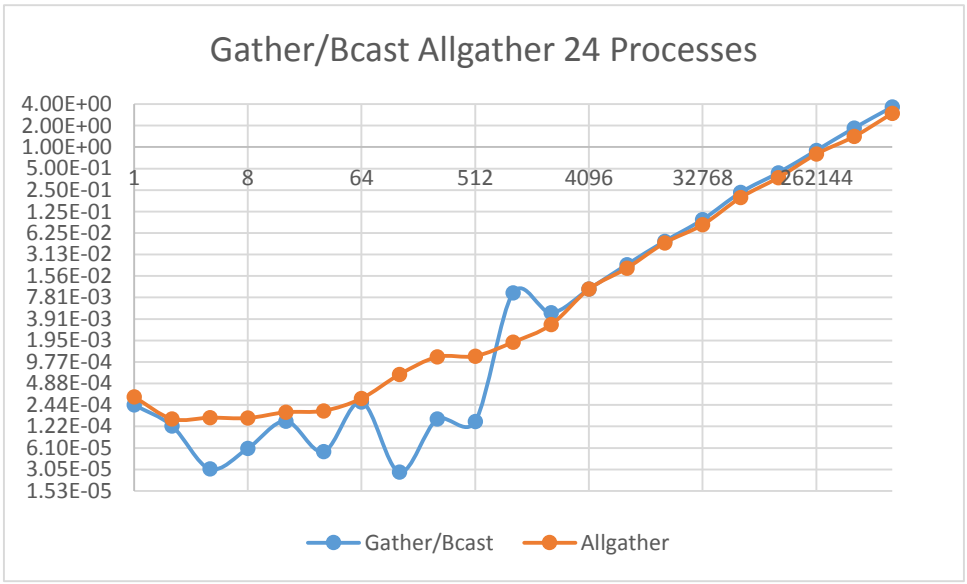
b.





CS4234 P1
elobeto
Evan Lobeto

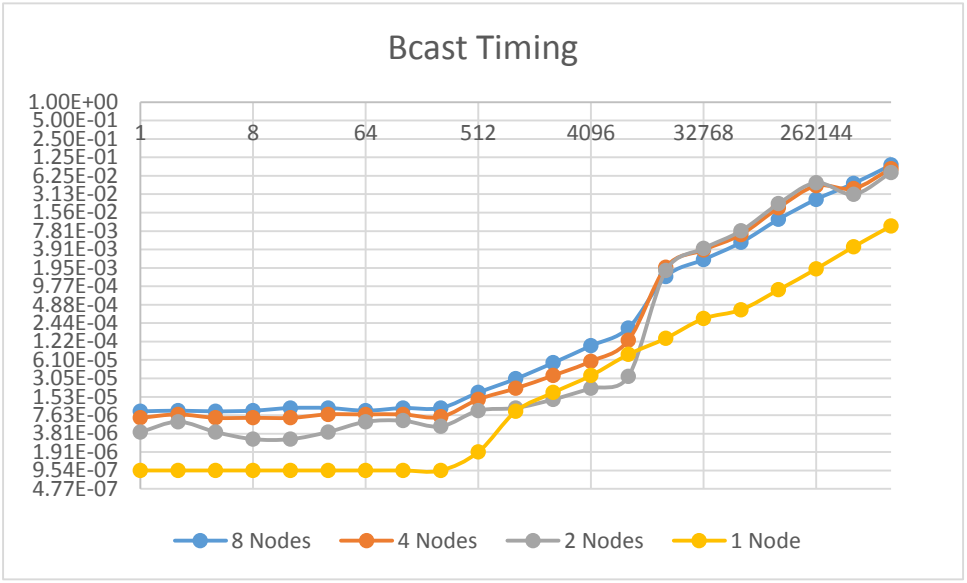




From the charts above, the same conclusion can be drawn as in question 2a. Comparing 2a and 2b, however, shows us that whether or not the data is summed at the end of gathering does not have a noticeable effect on the time it takes to complete. This probably occurs because of the fact that most of the time is taken in sending and receiving the data and the summation is run in parallel on each node and is a simple task compared to moving data around.

CS4234 P1
elobeto
Evan Lobeto

c.



From this graph it can be seen that using MPI_Bcast to send various data sizes between cores in a single processor takes significantly less time than sending data between nodes. This is most likely because of the network latency involved; a single processor can just share data from its memory rather than sending data across a network. Also, the time remains relatively constant until a size of 512 where it starts to increase steadily and then jump up in time at a size of 8192. This probably occurs because of buffer sizes inside the processor and in the network as well.