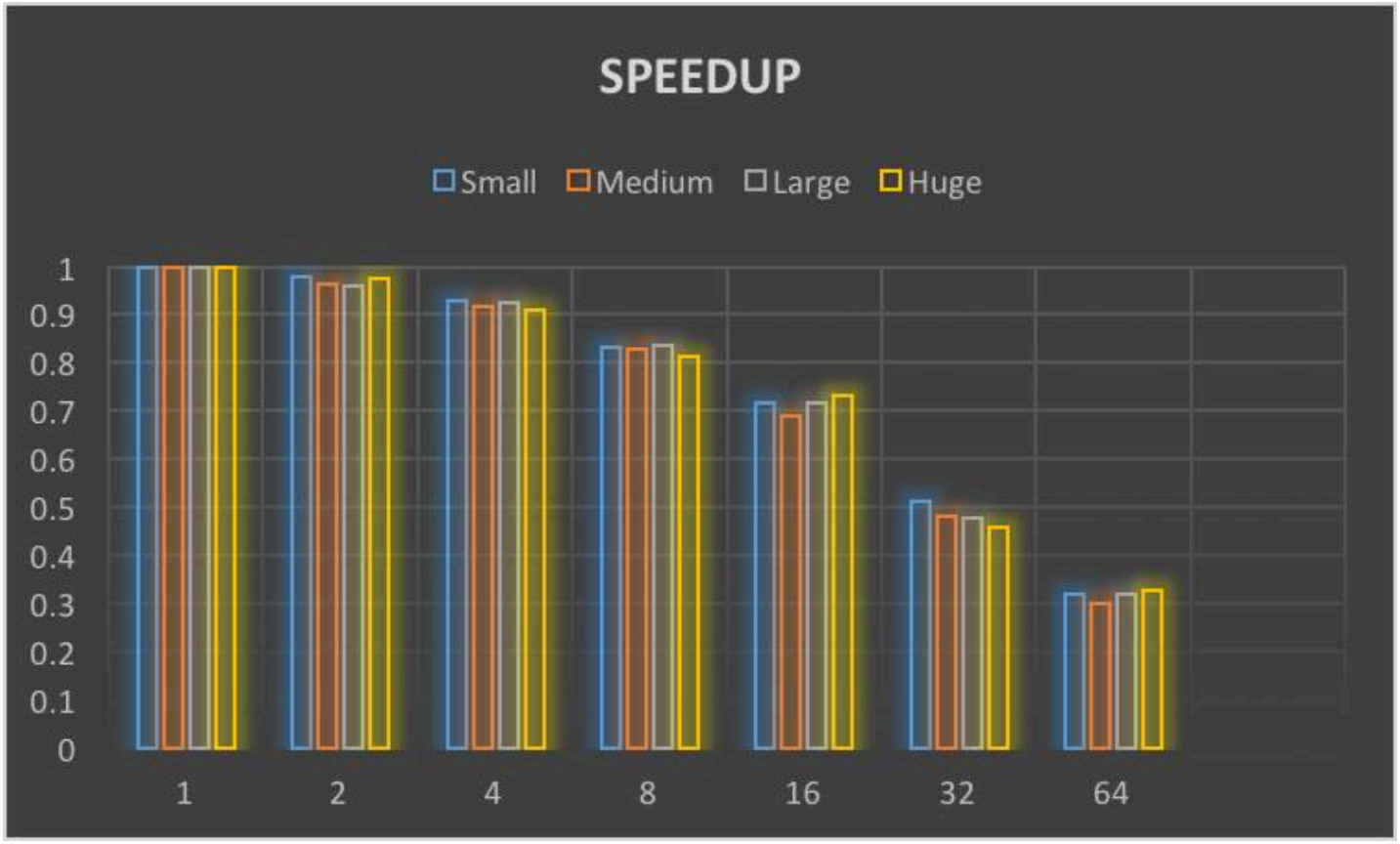
Jessica Donahue
Parallel Computing
3/21/16

Lab 1 Report



Conclusions:

From the bar graph, it is clear that the speedup decreases as we increase the number of processes, as a result of the increasing running times. For the four different data sizes, the graph has this similar decreasing pattern. This is because as we increase the number of processes, the amount of required communication among the processes increases, which is greatly taxing on the performance. For every iteration in the while loop, every process calls MPI_Allgatherv, which collects all the new local x values assigned to that process into an array. MPI_Allgatherv is a collective call, meaning it involves every process in the communicator to communicate. This call gathers data from all the processes and then delivers the combined data, the new x value after the computation, to all tasks. Thus, it is clear that as we increase the amount of processes, this collective call will take longer time to gather all the new values.

Also, when the number of processes becomes greater than the number of unknowns (for example, 3 unknowns in small.txt and 64 processes) then it is clear why there is no

speedup for these cases. If the number of unknowns is less than the number of processes, the program creates these extra processes and then terminates them because they aren't needed. The program is not using every process efficiently because not every process can be assigned an x value to evaluate – this decreases the performance. On the other hand, we can see from the graph that the programs have better "speedups" when there are fewer processes. This is because the overheard from communication greatly decreases the performance, thus when we run the program with less processes, there is less communication and better performance. From this lab, it is clear that parallelism is not always more efficient in terms of performance. If I designed the program to use less communication, I might have seen better optimization of parallelism.