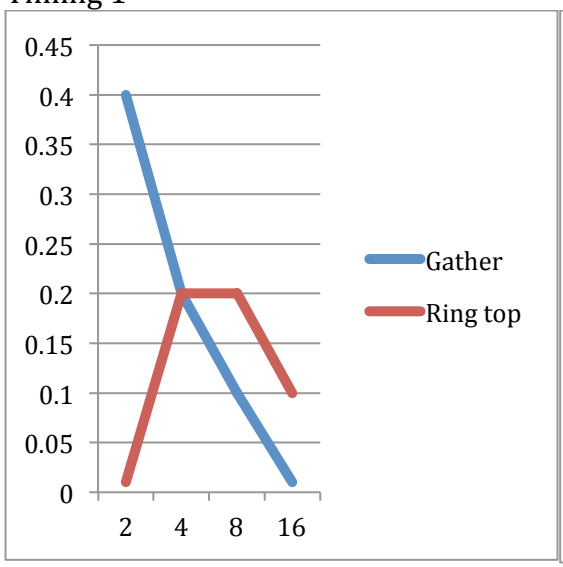
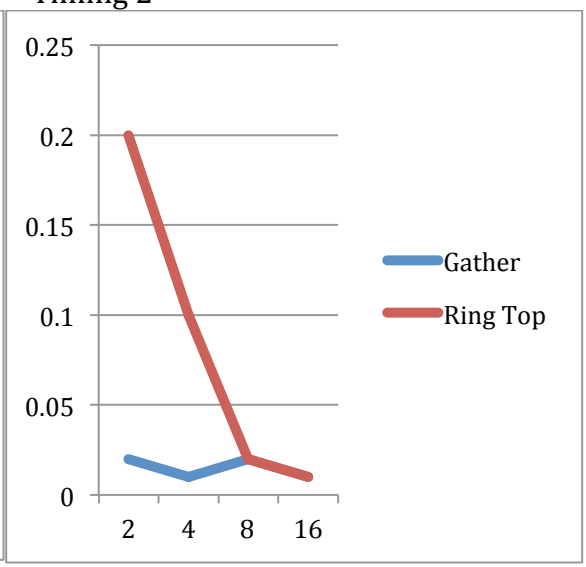
Ian Schweer
22514022.
Project 2
GRAPHS:

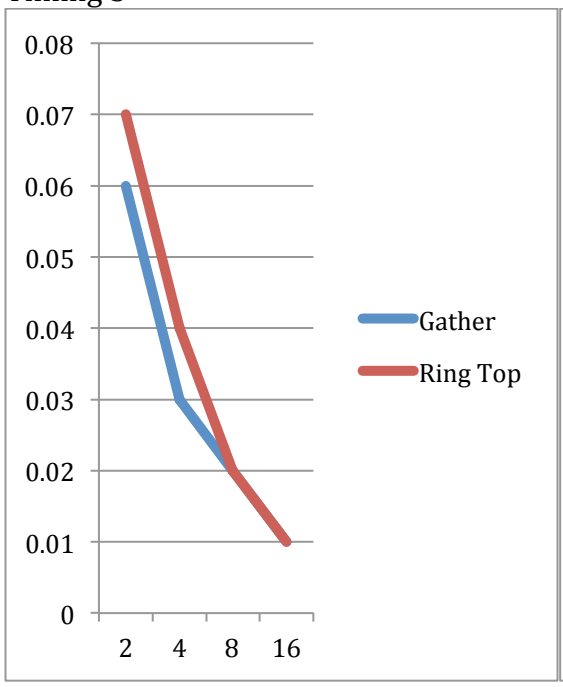
Timing 1



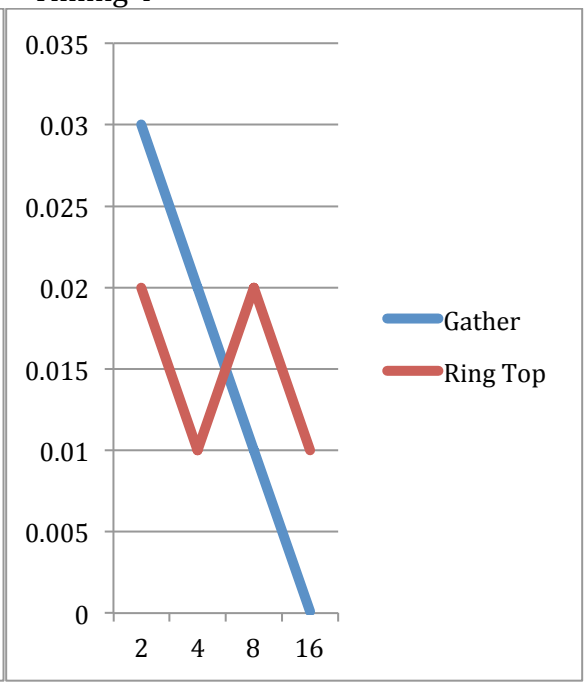
Timing 2



Timing 3



Timing 4



Analysis

So we can see that in general that as we add more processes we see an incremental speed up. As opposed to adding threads, we don't have to worry about sharing memory on the same machine; instead we let MPI do all the work for us. However we can see that the file does affect the results of the performance. Each file

has strings that are only length of 15 characters, so finding a palindrome is quick regardless of algorithm. A slight optimization is skipping all blank lines. Timing 1 contains roughly 90000 empty strings. Timing 2 contains roughly 91000. Timing 3 contains substantially less: 7327 and timing four contains 99788. We can see in general (by the y axis) as the number of white spaces goes down, we see an increase in performance.

Similar to threads, adding more increases our performance. Although this isn't entirely true because setup time takes longer then spawning more threads. The number of machines does affect our communication however. When using MPI_Gather, all slave machines will communicate with the master node, and the master node just sorts the data by rank. Our custom gather on the other hand forces each process to wait for the previous to finish. Explains why when there are more processes, the ring topology is slower then MPI gather. In our case though, a results struct is tiny (4 bytes per value * 3 values = 12 bytes / result). This is relatively fast to copy and sort, so the difference is minor. If this result were larger, then we would really see a difference. The ring topology is not as scalable as MPI_Gather's topology.