1. There are several, one is that each lookup of a query or point is on the heap. This means that this access is not fast at all. Another is that each of these query look ups happen N times, not just once.

# of Ranks	Time (s)	Speedup	Parallel Efficiency	Global Sum	Job Script
1	838.960602	N/A	N/A	96192511	Act1/1.sh
4	202.825502	4.13636645159	1.03409161289	96200864	Act1/4.sh
8	101.894030	8.23365806613	1.02920725826	95623343	Act1/8.sh
12	68.320090	12.2798521196	1.02332100996	96355910	Act1/12.sh
16	51.347655	16.3388299231	1.02117687019	96266230	Act1/16.sh
20	41.282907	20.3222268722	1.01611134361	96050298	Act1/20.sh

2. This algorithm does scale very well, reaching an efficiency of above one at every step. This speedup could continue for a while, although not indefinitely, this depends on the time that is inevitable.

# of Ranks	Total Time	Tree Time (s)	Search Time(s)	Global Sum	Job Script
1	9.784525	6.277307	3.507218	96192511	Act2/1.sh
4	7.469314	6.569332	0.899982	96200864	Act2/4.sh
8	8.108501	7.673479	0.435022	95623343	Act2/8.sh
12	7.973408	7.678568	0.294840	96355910	Act2/12.sh
16	7.656939	7.369366	0.287573	96266230	Act2/16.sh
20	8.220014	8.009489	0.210525	96050298	Act2/20.sh

# Ranks	Speedup	Parallel Efficiency
1	N/A	N/A
4	3.89698682	0.97424670
8	8.06216237	1.00777029
12	11.8953262	0.99127718
16	12.1959224	0.76224515
20	16.6593896	0.83296948

3. This tree construction time increases and this is likely just due to the time it takes for all ranks to complete this construction. The time should really not be affected so much but there are differences in the times ranks take to spin up, or just general execution speed differences that are more likely to see in when there are more cores.

4. The R-Tree has significantly better performance. The decreases in lookups and brute force means the algorithm takes only a fraciton of the time the brute force algorithm takes.

5. The brute force algorithm has better efficiency. This difference is due to the way these algorithms work. Decreasing Q for the brute force algorithm has a significant impact, while the impact is less for the R-Tree algorithm.

# Ranks	Total Time (s)	R-Tree Construction	Search Time (s)	Global Sum	Job Script
1	9.77346	6.276196	3.497264	96192511	Act3/1.sh
4	7.4605879999	6.556787	0.903801	96200864	Act3/4.sh
8	7.4827379999	6.962248	0.520490	95623343	Act3/8.sh
12	7.548529	7.188449	0.360080	96355910	Act3/12.sh
16	7.653466	7.370074	0.283392	96266230	Act3/16.sh
20	7.795418	7.556755	0.238663	96050298	Act3/20.sh

# Ranks	Speedup	Efficiency
1	N/A	N/A
4	3.8695066723	0.96737666809
8	6.71917616092	0.83989702011
12	9.71246389691	0.80937199140
16	12.3407294489	0.77129559056
20	14.6535659067	0.73267829533

6. Speedup and efficiency are both generally worse on two nodes. This is likely due to the network communication although there isn't much of this at all. While there is a noticable difference between the two (except on 16 cores), it is not particularly interesting.

7. The only parameter I changed was the explicit partitioning of ranks on each node. This was to see what kind of effect load balancing between nodes has.

# Ranks	Total Time (s)	R-Tree Construction	Search Time (s)	Global Sum	Job Script
1	10.189781	7.072190	3.117591	96192511	Act4/1.sh
4	7.50951	6.596254	0.913256	96200864	Act4/4.sh
8	7.451833	6.950113	0.501720	95623343	Act4/8.sh
12	7.540832	7.187828	0.353004	96355910	Act4/12.sh
16	7.656189	7.362008	0.294181	96266230	Act4/16.sh
20	7.736093	7.503582	0.232511	96050298	Act4/20.sh

# Ranks	Speedup	Efficiency
1	N/A	N/A
4	3.41370984696	0.85342746174
8	6.21380650562	0.77672581320
12	8.83160247475	0.73596687289
16	10.5975266927	0.66234541829
20	13.4083591744	0.67041795872

8. My new experiment worked significantly worse than the 2 node experiment, which I was not expecting. I expected that since each rank works fairly individually, that taking the first available cores might help things to execute more quickly but this was not the case.

9. I would rather the other user be running a compute bound algorithm because if their data was flagged explicit it would be very difficult to use the memory required, but CPUs spend a lot of their time idle and CPU use is more readily available than blocked off memory. This would also depend on the type of program I was running though.