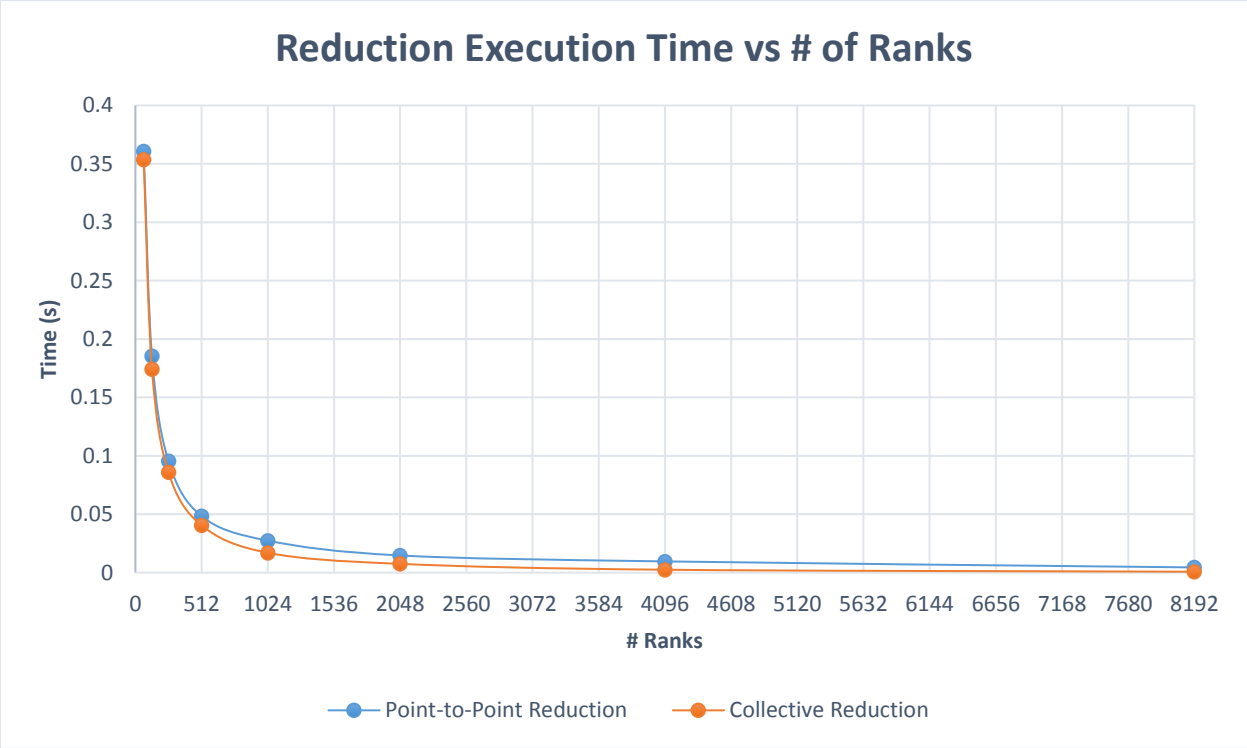
Ryan Stillings

Parallel Programming and Computing

Assignment 3 Performance Report

Performance Graph



Raw Data

# Ranks	Point-to-Point Execution Time (s)	Collective Execution Time (s)
64	0.3608	0.353551
128	0.185336	0.174158
256	0.095646	0.085956
512	0.048426	0.040551
1024	0.02744	0.017053
2048	0.014756	0.007545
4096	0.009664	0.002525
8192	0.004529	0.00082

Analysis

First of all, both P2P and Collective Reduction resulted in considerably faster performance each time the number of ranks was doubled. This was the expected outcome; since both reduction techniques split their work equally across all available ranks, the addition of more ranks should result in the runtime being reduced proportionally. Also as expected, the execution time for both algorithms was roughly halved each time the number of ranks was doubled. This was most likely due to the fact that the majority of the work being performed by each algorithm was in summing each individual rank's elements, an operation that is completed in linear time.

What was a bit surprising was the fact that the Collective reduction algorithm consistently took less time to complete than the P2P reduction algorithm. After some analysis, I noticed that the percentage difference between the Collective algorithm and the P2P algorithm increased considerably each time the number of ranks available was doubled. This increasing difference likely explains why the Collective algorithm was faster; regardless of how many ranks were used, the Collective operation needed only to have each rank send a single sum value to rank 0, who would then possess the final sum. The P2P algorithm, on the other hand, needed to send more sequential messages as the number of ranks used grew, increasing at a logarithmic rate. This, along with other minor factors such as more raw instructions being performed, is most likely the reason why the P2P algorithm performed worse than the Collective algorithm.