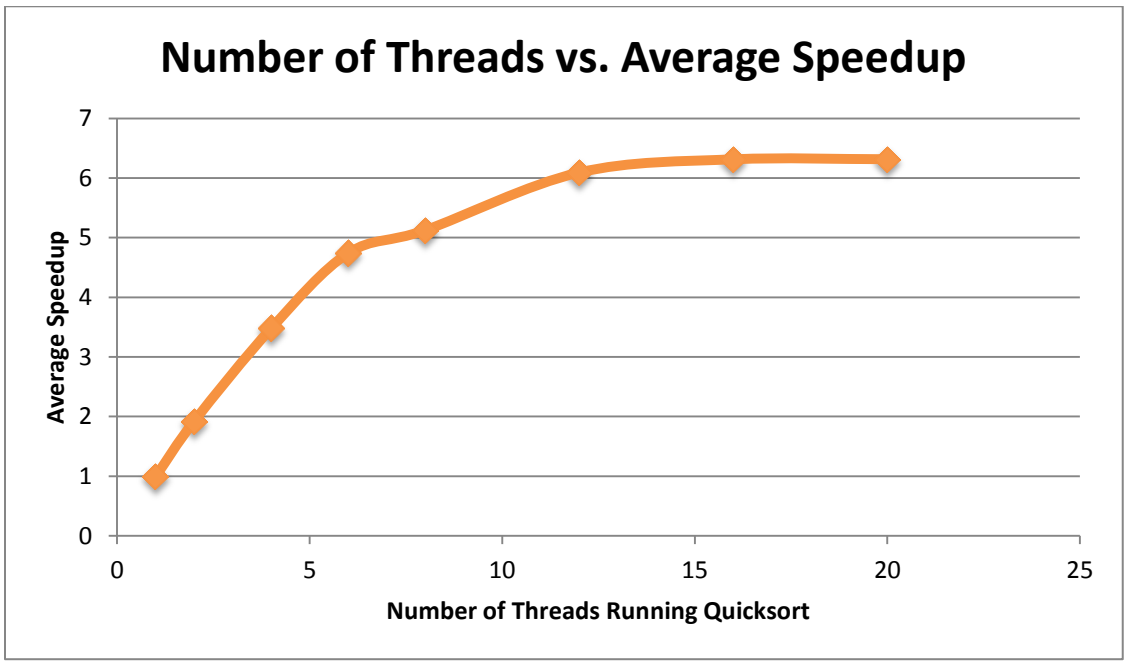
Thread Pool Discussion

Elmer Landaverde and Scott Sines

We ran the quicksort program, using our thread pool implementation, with 1, 2, 4, 6, 8, 12, 16, and 20 threads. Our results are displayed in the table and chart below.

Threads	Optimal Depth	Average Time	Base Time	Average Speedup
1	4	41.831	41.568	0.99
2	12	21.636	41.548	1.92
4	13	11.948	41.558	3.48
6	13	8.810	41.755	4.74
8	13	8.145	41.735	5.12
12	17	6.854	41.757	6.09
16	17	6.584	41.571	6.31
20	19	6.625	41.842	6.32



As expected, the average running time for the quicksort program decreases, and consequently the average speedup increases, as we utilize more threads. Therefore, we conclude that quicksort does scale well, up to a certain point, using this implementation on the multicore machines on the rlogin cluster. Since quicksort is a divide-and-conquer sorting algorithm, utilizing more threads allows us to farm out the work to more and more processing entities. This implementation of concurrent programming allows us to significantly reduce the running time of the program. Looking at the specs, the machines on the rlogin cluster have 16 usable threads. Looking at our data, the speedup of 16 threads and 20 threads are nearly identical showing that the scaling only has a significant effect up to 16 threads, which makes sense based on the rlogin machine specs.