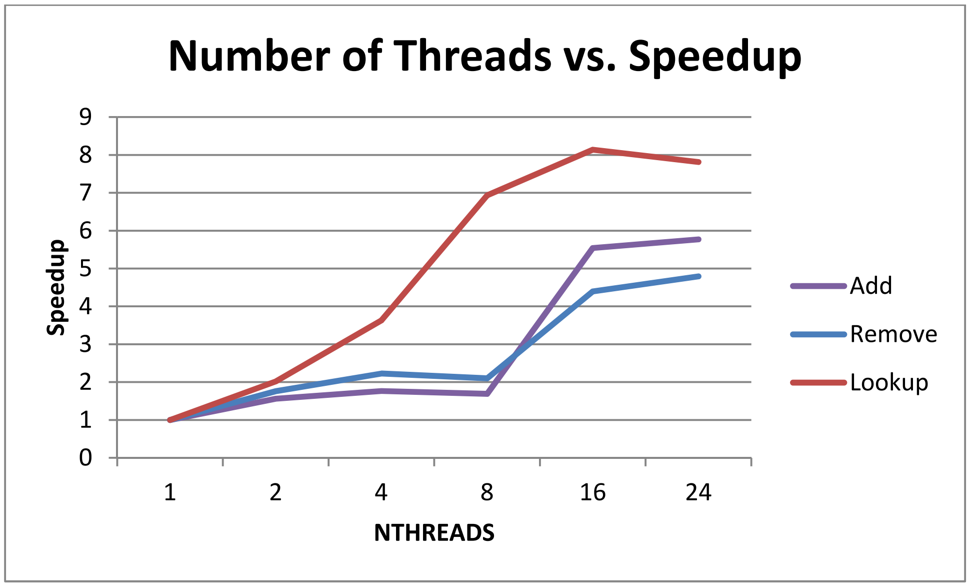
Performance Report
Concurrent Hash Map

I. Number of Threads vs. Speedup

The following chart shows number of threads versus speedup factor for add, remove and lookup operations averaged over 3 runs on spice-rack for 1, 2, 4, 8, 16, 24 threads. There were 100,000 of each lookups, adds, and removes for each trial. All data was randomly generated at the start of each trial, but the random number generator was given the same seed when running on different thread counts for consistency.



	speedups		
threads	add	remove	lookup
1	1	1	1
2	1.56238916	1.761033806	2.017465107
4	1.765914096	2.223227773	3.631429744
8	1.685959247	2.096925472	6.934384191
16	5.5423795	4.395652003	8.138959666
24	5.770271797	4.794129535	7.813132591

The times for the operations actually varied rather widely on different trials (where the random seed did vary amongst trials of the same thread count), and this is due to the randomness in our addition and removal of elements.

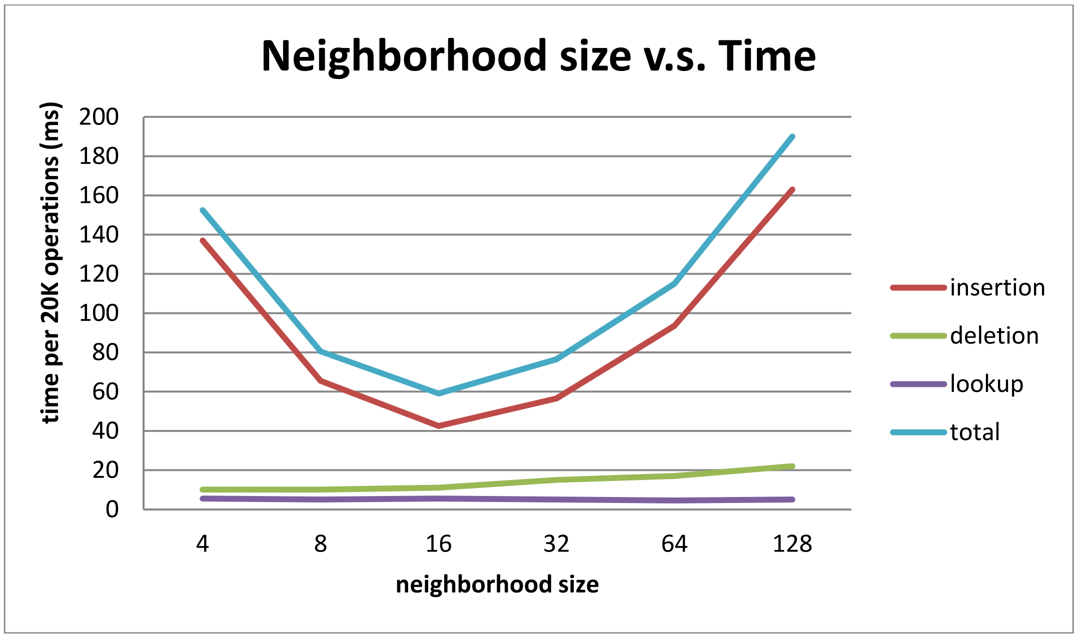
Overall there were decent speedups in lookup with a total speedup of about 8x with 24 threads (though varying depending on the run). Add and remove have worse scaling but still showed speedups of around 6x for our trials.

II. Neighborhood size vs. time per operation

Deletion:
From the plot, we can tell that the computation time of deletion is nearly irrelevant to the size of neighborhood size. Notice that deletion is supposedly slightly increasing with neighborhood size is because of the linear searching in neighborhood to find the key that is to be remove. The increase in deletion is more significant than in lookup. The reason is, as described above, that lookup uses bitmap instead of searching through the whole neighborhood. The plot shows the difference between deletion and lookup.

Lookup:
Lookup searches through neighborhood to find the key, same as what deletion does. However, the difference is that lookup checks the bitmap instead of searching in the buckets directly. The data structure of bitmap now is a sparse array, which might be the reason why lookup should slightly increase over neighborhood size. However in this plot we do not observe that. Lookup is nearly irrelevant to neighborhood size in this plot.

Insertion:
We can tell that the insertion dominates the computation time. Computation time of insertion increases over neighborhood size after some point and is decreasing over neighborhood size before some point. That is to say, there exists a minimum computation time at some optimal neighborhood size. In this case at neighborhood size equal to 16, the computation time reaches its minimum. The optimal neighborhood size is supposedly majorly related to the number of buckets, which in this case is 20,000/0.75(desired load factor) = 26,666



		insertion	deletion	lookup	total
number of operations		20000	10000	20000	50000
neighborhood size	4	274	10	11	295
	8	131	10	10	151
	16	85	11	11	107
	32	113	15	10	138
	64	187	17	9	213
	128	326	22	10	358