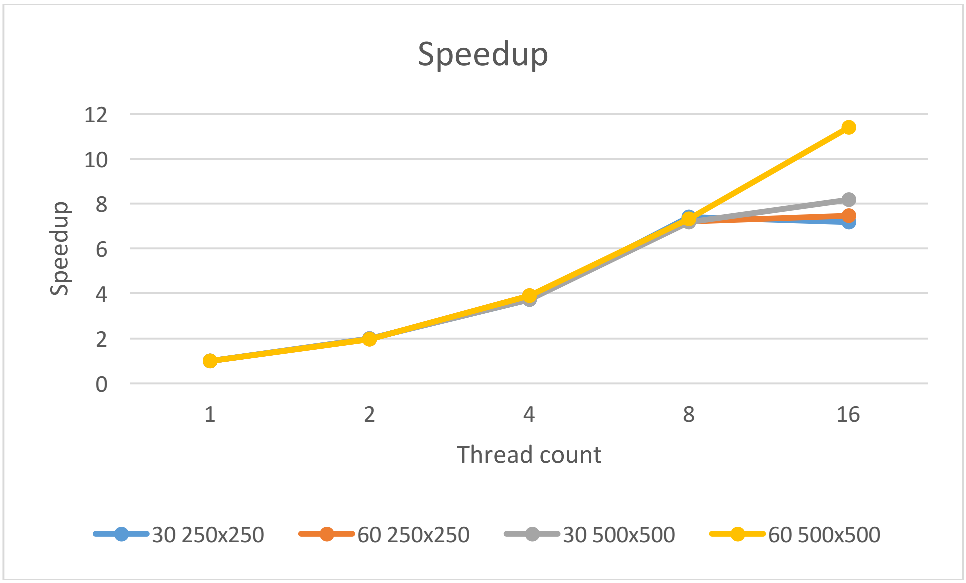
Diego Loya
Project 4
Parallel programming

4.1) Fractal

		Runtimes (secs)				
		1thread	**2threads**	**4threads**	**8threads**	**16threads**
30	**250x250**	1.04	0.52	0.28	0.14	0.14
60	**250x250**	2.03	1.04	0.52	0.28	0.27
30	**500x500**	4.13	2.08	1.11	0.58	0.51
60	**500x500**	8.12	4.14	2.08	1.11	0.71



The program's scalability drops as the thread count increases. When there is no schedule assigned, the default schedule is block. This causes a load imbalance, which is the reason for the scalability drop.

4.2) TSP

Runtimes (secs)

	CPU	**MIC**
1291	1.29	0.90
1379	3.31	2.45
1400	2.85	2.12

The runtime depends on the input because of the do-while loop, which increases runtime considerably as the number of iteration increases. The increase in performance is due to the SMT technique, which allows multiple independent threads to be executed simultaneously. Some performance is lost because of thread creation.
There is a speed up of roughly 1.35 when using MIC chip. Though it does not scale as expected, given the considerable difference in threads and the fact that each thread is independent and runs simultaneously.

A named critical section is only needed when two blocks of code will be executed simultaneously. In our case, the threads enter this part sequentially because they depend on each other's result.

Duplicating the if statement prevents from going into a critical section when not needed, which will improve the performance.

4.3) BST

Runtimes (secs)

	Thread count				
	1	**2**	**4**	**8**	**16**
100000	0.0393	0.0282	0.016	0.0103	0.0175
1000000	0.6321	0.391	0.2089	0.1263	0.1621
10000000	12.2928	6.8858	3.549	2.0038	1.2785

The default schedule works as a block schedule. The reason why performance drops as threads reach 8 and 16 threads might the load imbalance caused by default scheduling. Also, as more threads are created, more locks are necessary, leading to lower efficiency.

A root node needs to be created. If the loop started at 0, then there would be no root to start at and therefore our code would fail.

The hash function is re-entrant because it is called by different threads with different values, but same seed. For this reason, it does not affect any other thread.

Only the parent node needs to be locked because other nodes will not be modified, only read. A node only needs to be locked when writing to its left and right nodes, while other threads are safe to traverse.

The reason we recheck is that another thread might have been faster and modified that node before we acquired the lock. This guarantees thread safety for our program.