Exercise 1

My implementation for the blocking concurrent queue is quite straightforward. I use a mutex to lock at the begin of both push and pop, and release the mutex before the return of push and pop. And it passed my tests of enqueuing and dequeuing of more than 100000 nodes from multiple threads.

Exercise 2

The lock-free concurrent queue I implemented is the Michael and Scott queue from the textbook--"Shared Memory Synchronization". And it passed my tests of enqueuing and dequeuing of more than 100000 nodes from multiple threads.

Exercise 3

1. Performance report for "sssp_blocking" with all kinds of graph and thread number of 1, 2 ,4, 8, and 56 on node2x14, which has 56 hardware threads.

simplegr_blocking

n_thread	mean	std
1	0	0
2	0	0
4	0	0
8	0	0
56	2.45	0.668954408

NY_blocking

n_thread	mean	std
1	1922.5	36.36963019
2	8198.55	1061.367819
4	8566	1112.278832
8	10908.5	1153.133969
56	14697.85	531.361673

Rmat_16_blocking

n_thread	mean	std
1	44.85	1.194780315
2	140.7	16.96201639
4	162.85	25.87136448
8	176.8	36.82607772
56	211.45	5.333619784

Rmat_22_blocking

n_thread	mean	std
1	6348.65	457.0487146
2	7959.4	953.3536804
4	8633.65	822.9989839
8	10620.75	1534.470947
56	12165.15	2428.546485

2. Performance report for “sssp_lockfree” with all kinds of graph and thread number of 1, 2 ,4, 8, and 56 on node2x14, which has 56 hardware threads.

simplegr_lockfreee

n_thread	mean	std
1	0	0
2	0	0
4	0	0
8	0	0
56	2.55	0.589491306

NY_lockfreee

n_thread	mean	std
1	4355.3	156.2632714
2	5480.2	1944.364539
4	8935	2890.035329
8	15797.35	4697.445117
56	59885.1	3718.508813

Rmat_16_lockfree

n_thread	mean	std
1	74.75	3.561951712
2	78.1	4.742362281
4	122.1	45.32317288
8	315.65	65.19070102
56	933.5	216.2839569

Rmat_22_lockfree

n_thread	mean	std
1	10990.9	958.3902598
2	10797.95	1200.894686
4	8331.9	865.5356665
8	11931.05	3304.21878
56	61771.3	9292.641756

3. My implementation of lock-free queue (Michael and Scott queue) performs much slower than the blocked concurrent queue that use one mutex per pop/push. And there are two ways to optimize the lock-free queue that I think could help a lot. First, the way I solve the ABA problem is by counting the reference, and every compare and swap in enqueuer and dequeuer allocate a new “ptr” that contains a incremented counter, so it not only has more computation for the counter but also has more memory usage. Second, every push allocate memory for the new node, so it’s also another overhead that the blocked concurrent queue does not have (allocate memory that it could use when initializing it).