Q1.6

The graph above describes the relationship between computational time and the number of threads used to run 2000x2000 matrix multiplications in parallel. Note this program was run on a quad-core processor. As shown, the time for each computation decreases as the number of cores used increases. However a plateau begins at around 5 cores where the computational time difference is negligible. This is likely due to the fact that up to 4 threads may actually run concurrently (one thread on each core), and running more than 4 threads would delegate more than one thread to some (or all) cores and thus mimic sequential computation, and would therefore not increase the speed .of computation

The graph above describes the computational time relationship with the size of the matrices being multiplied, computed both sequentially (in blue) and in parallel (in red, with 9 threads). As shown, the computational time for the matrix multiplication of smaller matrices (up to 1000x1000 in this case) run similarly. This is likely due to the overhead of running the program in parallel, and thus for smaller matrices sequential computation will be similar. As the size of the matrices continue to increase however, the sequential computation begins to slow down and the use of multiple cores has a greater effect.

Q2

2.1:

Deadlock may occur when two or more threads are attempting to acquire the same locks on shared objects. Deadlock describes the situation where each thread is waiting for a lock that another has, and thus no progress is made

2.2:

There is a technique known as resource ordering that would prevent deadlock from occurring. This technique assigns an order to all objects whose locks are required, where each thread acquires each lock in that order. This way no two threads will hold each other’s locks due to the intentional ordering, as if each thread waits in a queue for the next resource and may only obtain the resources in that order, thus allowing the program to eventually terminate.