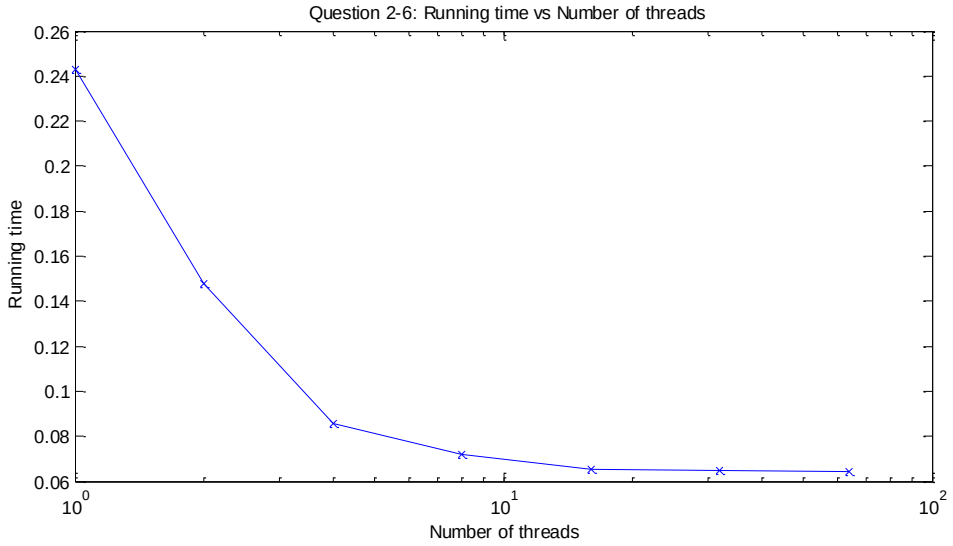
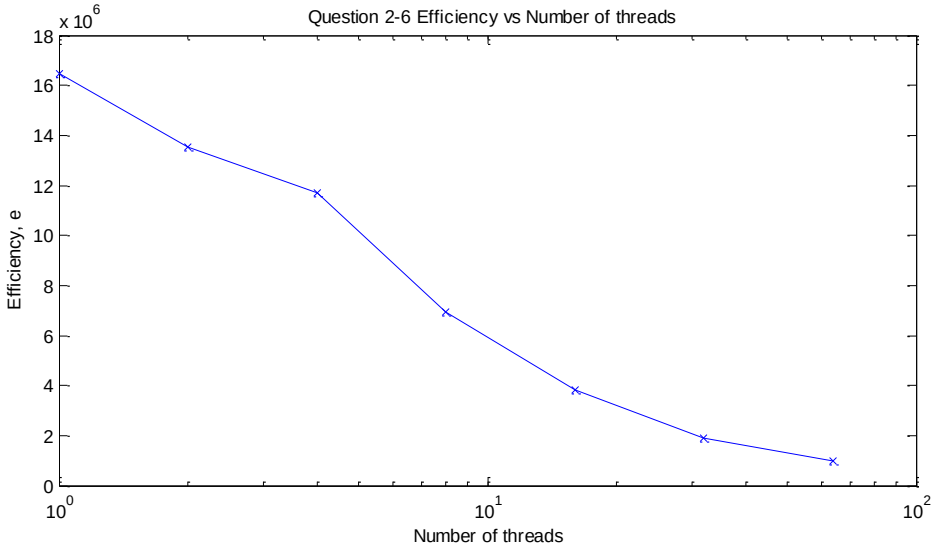
HW1 Writeup

Problem 2

Question 6

The plots of running time and efficiency against number of threads are shown below:





It can be seen from the first plot that as the number of threads increases from 1 to 16, the running time decreases sharply. However, the curve flattens as the number of threads further increases. It can also be seen from the second plot that as the number of threads increases, the efficiency decreases. Ideally, the efficiency should be constant. The efficiency decreases because not every part of the algorithm is parallelized. Some functions in the code are still sequential like reduceLocalHistoToGlobal() and computeBlockExScanFromGlobalHisto(). Besides,

there is certain amount of overhead when using threading. Therefore, increasing the number of threads cannot increase the running speed ideally.

Question 7

To parallelize the function reduceLocalHistoToGlobal(), the code is parallelized over buckets instead of over blocks. By looping over buckets and add value to globalHistro, we won't have the race condition in the case if we loop over the blocks. The code is shown below:

```
std::vector<uint> reduceLocalHistoToGlobal(const std::vector<uint>&
blockHistograms, uint numBlocks, uint numBuckets) {
    std::vector<uint> globalHisto(numBuckets, 0);

#pragma omp parallel for
    for (uint i = 0; i < numBuckets; i++)
    {
        uint sum = 0;
        for (uint n = 0; n < numBlocks; n++)
        {
            sum += blockHistograms[n*numBuckets + i];
        }
        globalHisto[i] = sum;
    }
    return globalHisto;
}
```

Question 8

Like the previous part, to parallelize the function computeBlockExScanFromGlobalHisto (), the code is parallelized over buckets instead of over blocks. The code is shown below:

```
std::vector<uint> computeBlockExScanFromGlobalHisto(uint numBuckets, uint
numBlocks, const std::vector<uint>& globalHistoExScan, const std::vector<uint>&
blockHistograms) {
    std::vector<uint> blockExScan(numBuckets * numBlocks, 0);
    std::vector<uint> tempHisto(numBuckets, 0);

#pragma omp parallel for
    for (uint i = 0; i < numBuckets; i++)
    {
```

```
        for (uint n = 0; n < numBlocks; n++)
        {
            blockExScan[n*numBuckets + i] = globalHistoExScan[i] + tempHisto[i];
            tempHisto[i] += blockHistograms[n*numBuckets + i];
        }
    }

    return blockExScan;
}
```