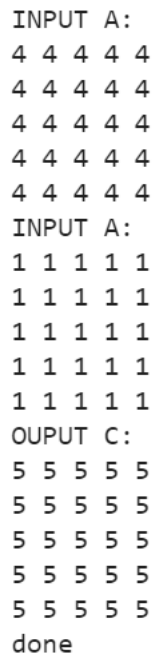
Theodore Jagodits / CS 677 / HW#1 / Report

1.



```
INPUT A:
4 4 4 4 4
4 4 4 4 4
4 4 4 4 4
4 4 4 4 4
4 4 4 4 4
INPUT A:
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
OUPUT C:
5 5 5 5 5
5 5 5 5 5
5 5 5 5 5
5 5 5 5 5
5 5 5 5 5
done
```

4.

There will be 4*512(2048) threads in the grid. Because 512 threads in one block so you must allocate that amount that will fit 2,000.

For the second question, there will only be one warp that will have divergence. So, there will be 2,048 threads allocated for the kernel. But only 2,000 of them will be active. Looking at the last few threads:

2,048 – 2,000 = 48 threads. One entire warp will be inactive: 48 – 32 = 16 threads. This last warp has 16 active threads and 16 inactive threads causing divergence. We don't consider the inactive warp as divergent since **all** threads are inactive so no divergence.

5.

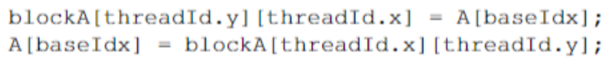
For matrix addition, when each element is accessing one corresponding index there is no use for shared memory. The threads share no common elements so putting the matrix memory from global into shared is just one extra step taking time away from the program. This is because shared memory is only useful when multiple threads are accessing the same memory in a block. However, in matrix addition each element accesses its own unique corresponding data point so there is no point of putting it into shared from global and then the thread accessing it from there.

6.

A) The block width (BW) is valid for the values of 1 – 5, 6 and above it runs into issues. First any number above 5 squared is above 32(6^2 =36….). Which exceeds the size of a warp. The problem here is when you try to transpose you need to keep all the values without overwriting them when your transposing with this tile method. When the BW*BW (matrix size is BW*BW) is lower than warp size this is fine for tiles since all threads finish within a warp at the same time. However, if the BW*BW is greater than a warp, there will be multiple warps doing the transpose which may not finish at the same time which runs into concurrency problems. Since some values might be already copied over and overwrite and finish execution at different times.

B)

To solve this put a thread sync command between these lines:



```
blockA[threadId.y][threadId.x] = A[baseIdx];
A[baseIdx] = blockA[threadId.x][threadId.y];
```

This syncs the threads across the kernel as it moves on to the next transpose step so that it cannot overwrite some parts of its own matrix.