**EC 527 Lab 1**

**Part 1)**

Q1- Problems start to occur when Delta is roughly 2000. The execution problems occur because the program couldn’t allocate storage.

#define BASE 0

#define ITERS 30

#define DELTA 200

#define BASE 0

#define ITERS 30

#define DELTA 400

#define BASE 0

#define ITERS 30

#define DELTA 600

#define BASE 0

#define ITERS 30

#define DELTA 800

#define BASE 0

#define ITERS 30

#define DELTA 1000

Combine2D is faster by roughly 1.6 times. At array sizes greater than 6000, there is a lot of sudden changes of slope, or spikes in the number of cycles for combine2D\_rev. These spikes and drops may be occurring due to spatial or temporal locality issues.

Q 2) Adjusted:

#define BASE 1000

#define ITERS 2

#define DELTA 5000

6000, 106951632, 186516172

11000, 359787296, 628536572

Cycles per element for combine2D = 2.97 when matrix width is 11000, whereas for combine2D\_rev = 5.19 for the same width. Combine2D\_rev has a higher CPE, since in the for loop in combine2d\_rev the function has to change the variable for the multiplicand(j), every iteration, which is a more costly operation than addition. Since the additive term is what is changed every iteration in the regular combine2D function, this is more efficient.

Q 3) The cycles to element ratio (part d) may be improving for a short while because there is spatial or temporal locality taking place in the function. It may be spike up after the dip, as a new cache block is brought in from memory. The ratio for combined2D\_rev (part e) might be more jumpy owing to accessing memory more often than the regular combined2D function.

**Part 2**

1.

Ijk – There are 6 plateaus

Kij – There are 0 plateaus

Jki – There are 6 plateaus

2. Ijk – Point 180362125 (cycles per iteration roughly 4.93), point 10518751913 (cycles per iteration roughly 6.35) , Point 1920495907 (cycles per iteration roughly 6.65), Point 3170044709 (cycles per iteration roughly 6.86) , Point 7088952961(cycles per iteration roughly 7.91), Point 9896830523(cycles per iteration roughly 7.92).

JKi – Point 1051871913(cycles per iteration roughly 5.29) , Point 1920495907 (cycles per iteration roughly 5.72), Point 3170044709 (cycles per iteration roughly 5.86), Point 4869777375 (cycles per iteration roughly 5.94), Point 7088952961 (cycles per iteration roughly 7.08), Point 9896830523 (cycles per iteration roughly 6.76).

3. JKi -Transitions occur at Point 180362125, point 678, point 1017, point 1130, point 1243, point 1356, point 1469, point 1582, point 1921, point 2034, point 2147.

Ijk - Transitions occur at point 180362125, Point 10518751913, point 1130, point 1243, point 1356, point 1469, point 1582, point 1695, point 1808, point 1921, point 2034, point 2147.

1.

Ijk – There are 4 plateaus

Kij – There are 0 plateaus

Jki – There are 3 plateaus

2. Ijk – Point 4096000000 (cycles per iteration roughly 5.32), point 13824000000 (cycles per iteration roughly 5.26) , Point 21952000000 (cycles per iteration roughly 5.23), point 32768000000 (Cycles per iteration roughly 6.62), point 46656000000 (Cycles per iteration roughly 4.99)

JKi – Point 4096000000 (CPE roughly 8.57), Between 13824000000 and 17576000000 (cycles per iteration roughly 8.44) , Point 32768000000 (cycles per iteration roughly 11.5)

3. JKi -Transitions occur at Point 4096000000, point 13824000000, point 17576000000, point 32768000000

Ijk - Transitions occur at Point 4096000000, point 13824000000, point 21952000000, point 32768000000, point 46656000000

1.

Ijk – There are 3 plateaus

Kij – There are 0 plateaus

Jki – There are 2 plateaus

2. Ijk – Roughly at point 1157625000(cycles per iteration roughly 4.41), point 4096000000 (cycles per iteration roughly 5.28) , Point 7414875000 (cycles per iteration roughly 4.69

JKi – Point 1728000000(cycles per iteration roughly 7.59) , Point 4096000000(cycles per iteration roughly 8.56)

3. JKi -Transitions occur at Point 1728000000, point 4096000000

Ijk - Transitions occur at Point 1157625000, point 4096000000, Point 7414875000

**Part 3)**

Matrix size of

2400

Blocking is useful as it takes advantage of temporal locality of the inner loops. I took matrix size to be 2400, and varied the block size to check the time. After testing the variations of block size it can be seen that the optimal block size is roughly 20.

For the second trial, I took matrix size to be 6000, and varied the block size to check the time. As can be seen the optimal block size is roughly 20 again.

**Part 4)**

It can be seen that ij is faster than ji since ij takes in horizontal reads whereas ji takes vertical reads.

I also blocked my code, and set block sizes of 10, 60, and 110. I use four loops - two to iterate over the blocks, and then another two to perform the transpose-copy of a single block.

As can be seen the larger block sizes are faster than the smaller block sizes. The blocking version is more efficient than the non-blocking version, since in the blocking version, once a particular block is used, it is not referenced again, whereas in the non-blocking version, the same elements may be brought into the cache multiple times.

**Part 5)**

1. Roughly 6 hours total time
2. No part took an unreasonable amount of time
3. No skills were missing, except prior knowledge of what combine2d function is.
4. There are no issues with the lab