```
pollock:~/Desktop/cs475/proj/PA4/PA4$ vecMax00 32 32 1280000000
Result: 1280000000.000000
 Time to copy input: 0.618775
 Compute time: 0.575354
 Time to  copy output and generate final answer: 0.000043
pollock:~/Desktop/cs475/proj/PA4/PA4$ vecMax01 32 32 1280000000
Result: 1280000000.000000
 Time to copy input: 0.645172
 Compute time: 0.300824
 Time to  copy output and generate final answer: 0.000023
```

For the input size of 1.28 billion, vecMax01 gets close to a 2x speedup over vecMax00, and although it does not show the GflopsS, it is much higher for vacMax01 since it is exploiting locality better.

```
pollock:~/Desktop/cs475/proj/PA4/PA4$ matmult00 100
Data dimensions: 1600x1600
Grid Dimensions: 100x100
Block Dimensions: 16x16
Footprint Dimensions: 16x16
Time: 0.068930 (sec), nFlops: 8192000000, GFlopsS: 118.845359
pollock:~/Desktop/cs475/proj/PA4/PA4$ matmult01 50
Data dimensions: 1600x1600
Grid Dimensions: 50x50
Block Dimensions: 16x16
Footprint Dimensions: 32x32
Time: 0.012199 (sec), nFlops: 8192000000, GFlopsS: 671.521457
```

The optimized matmult gets a much better execution time, and a lot more GflopsS because it properly coalesces more memory accesses.

```
pollock:~/Desktop/cs475/proj/PA4/PA4$ matmult00 200
Data dimensions: 3200x3200
Grid Dimensions: 200x200
Block Dimensions: 16x16
Footprint Dimensions: 16x16
Time: 0.180826 (sec), nFlops: 65536000000, GFlopsS: 362.425860
pollock:~/Desktop/cs475/proj/PA4/PA4$ matmult01 100
Data dimensions: 3200x3200
Grid Dimensions: 100x100
Block Dimensions: 16x16
Footprint Dimensions: 32x32
Time: 0.218964 (sec), nFlops: 65536000000, GFlopsS: 299.300531
```

For this input size, matmult01 falls short. I believe this has to do with the footprint dimensions, because matmult00 did badly on the previous one where the input was 100x the footprint dimension, but I’m not sure what is going on here.

```
pollock:~/Desktop/cs475/proj/PA4/PA4$ matmult00 400
Data dimensions: 6400x6400
Grid Dimensions: 400x400
Block Dimensions: 16x16
Footprint Dimensions: 16x16
Time: 0.969880 (sec), nFlops: 524288000000, GFlopsS: 540.569909
pollock:~/Desktop/cs475/proj/PA4/PA4$ matmult01 200
Data dimensions: 6400x6400
Grid Dimensions: 200x200
```

```
Block Dimensions: 16x16
Footprint Dimensions: 32x32
Time: 0.419979 (sec), nFlops: 524288000000, GFlopsS: 1248.367605
```

Again, matmult is able to execute better with coalescing.

After playing with the block size, it seems that the performance will improve as long as all the shared data for all 3 matrices can fit nicely in shared memory, after that there are errors. Small block sizes don’t perform as well because they make more frequent memory accesses.

Some general tips on optimizing CUDA:

Coalesce data for improved locality and memory access.

Make sure that the memory banks don’t cause threads to wait sequentially to access an element in shared memory, by messing with the sizes or either blocks, column padding or any other valid method.