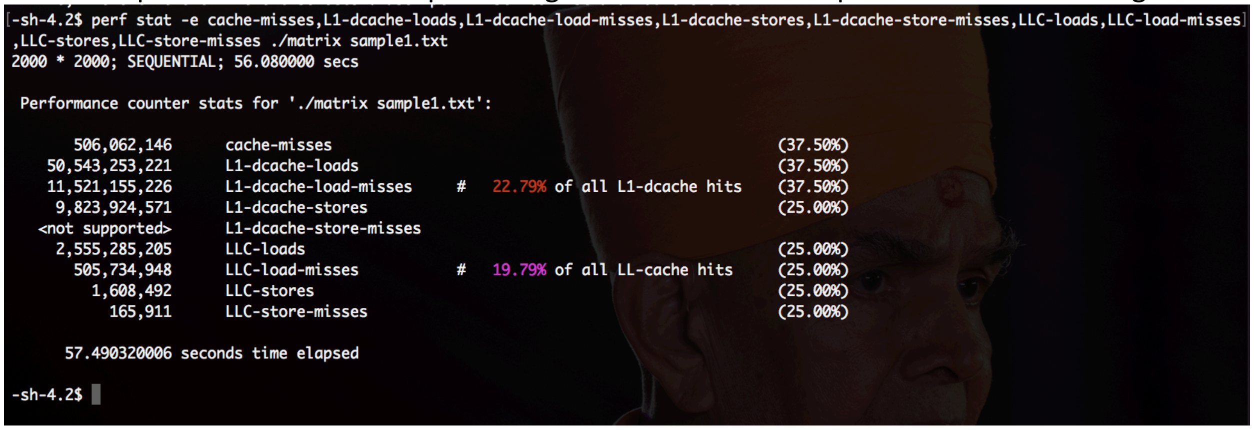
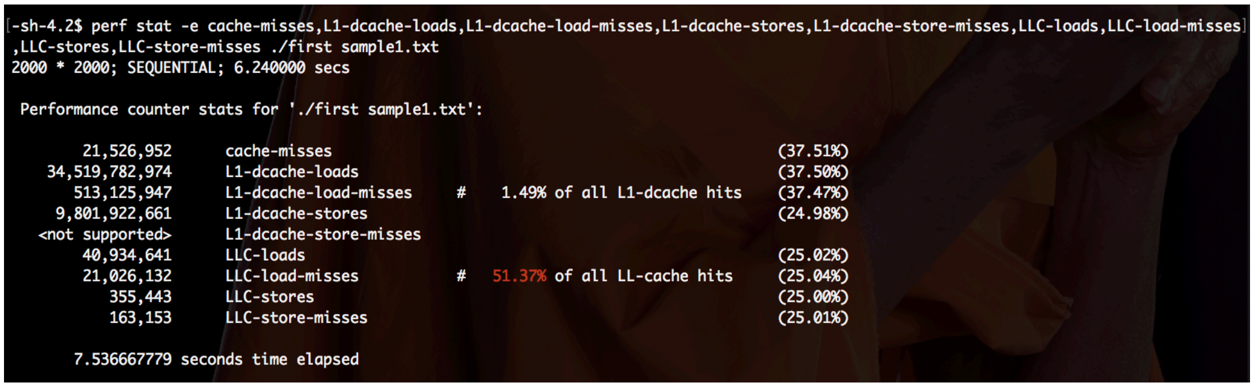
Part 1 Report:

The C code given by the professor will herein be referred as matrix.c, and my optimized code will be herein referred as first.c. In matrix.c the professor implemented a naïve algorithm without any cache optimization. In first.c, I have implemented Loop Fusion and Loop Interchange. I attempted Loop blocking however it is commented out and isn't running because the matrix_result.txt file did not match what it was supposed to me. Through Loop Fusion and Loop Interchange I was able to significantly improve cache performance. Referencing the photos below it can be seen that the average time between both programs was reduced from 55.977 to 6.307 Seconds that is a 88.73 percent improvement to the cache performance. Also the ratio between L1 cache misses to L1 cache hits was decreased from 22.79% to 1.49%. However for all lower level cache the ratio increased from 19.79% to 51.37%, this however still resulted in a greater increase in performance because the lower level caches are less quick. To further improve the LL cache loop blocking should have been implemented and working.



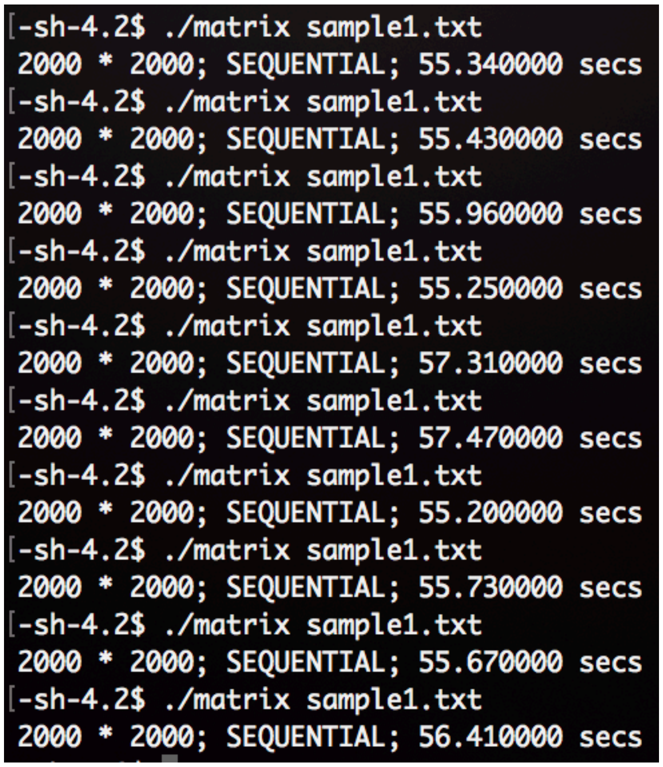
```
[-sh-4.2$ perf stat -e cache-misses,L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-dcache-store-misses,LLC-loads,LLC-load-misses]
,LLC-stores,LLC-store-misses ./matrix sample1.txt
2000 * 2000; SEQUENTIAL; 56.080000 secs

 Performance counter stats for './matrix sample1.txt':

       506,062,146      cache-misses                                                  (37.50%)
    50,543,253,221      L1-dcache-loads                                               (37.50%)
    11,521,155,226      L1-dcache-load-misses     #   22.79% of all L1-dcache hits    (37.50%)
     9,823,924,571      L1-dcache-stores                                              (25.00%)
   <not supported>      L1-dcache-store-misses
     2,555,285,205      LLC-loads                                                     (25.00%)
       505,734,948      LLC-load-misses           #   19.79% of all LL-cache hits     (25.00%)
         1,608,492      LLC-stores                                                    (25.00%)
           165,911      LLC-store-misses                                              (25.00%)

      57.490320006 seconds time elapsed

-sh-4.2$ 
```

^Cache Performance for matrix.c



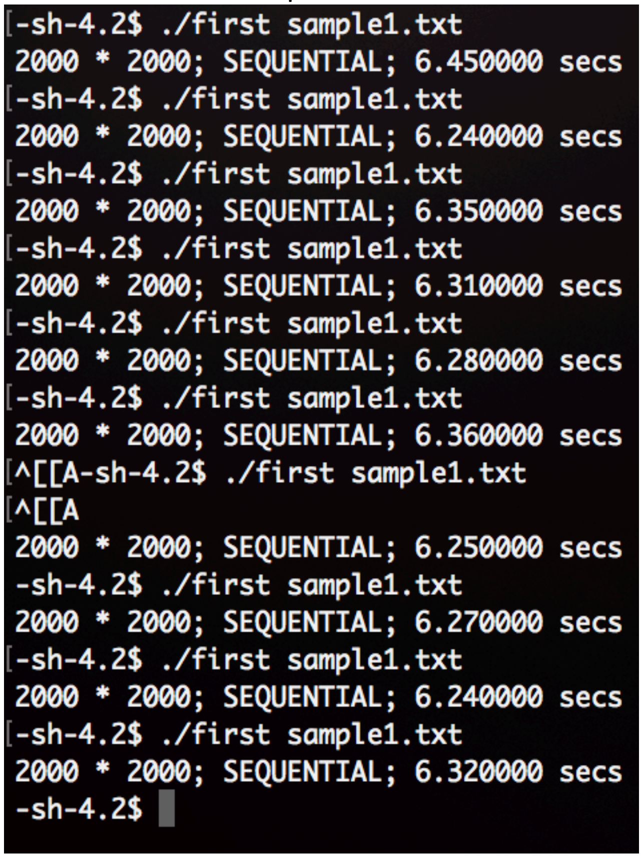
```
[-sh-4.2$ perf stat -e cache-misses,L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-dcache-store-misses,LLC-loads,LLC-load-misses]
,LLC-stores,LLC-store-misses ./first sample1.txt
2000 * 2000; SEQUENTIAL; 6.240000 secs

 Performance counter stats for './first sample1.txt':

        21,526,952      cache-misses                                                  (37.51%)
    34,519,782,974      L1-dcache-loads                                               (37.50%)
       513,125,947      L1-dcache-load-misses     #    1.49% of all L1-dcache hits    (37.47%)
     9,801,922,661      L1-dcache-stores                                              (24.98%)
   <not supported>      L1-dcache-store-misses
        40,934,641      LLC-loads                                                     (25.02%)
        21,026,132      LLC-load-misses           #   51.37% of all LL-cache hits     (25.04%)
           355,443      LLC-stores                                                    (25.00%)
           163,153      LLC-store-misses                                              (25.01%)

       7.536667779 seconds time elapsed
```

^Cache Performance for first.c



```
[-sh-4.2$ ./matrix sample1.txt
 2000 * 2000; SEQUENTIAL; 55.340000 secs
[-sh-4.2$ ./matrix sample1.txt
 2000 * 2000; SEQUENTIAL; 55.430000 secs
[-sh-4.2$ ./matrix sample1.txt
 2000 * 2000; SEQUENTIAL; 55.960000 secs
[-sh-4.2$ ./matrix sample1.txt
 2000 * 2000; SEQUENTIAL; 55.250000 secs
[-sh-4.2$ ./matrix sample1.txt
 2000 * 2000; SEQUENTIAL; 57.310000 secs
[-sh-4.2$ ./matrix sample1.txt
 2000 * 2000; SEQUENTIAL; 57.470000 secs
[-sh-4.2$ ./matrix sample1.txt
 2000 * 2000; SEQUENTIAL; 55.200000 secs
[-sh-4.2$ ./matrix sample1.txt
 2000 * 2000; SEQUENTIAL; 55.730000 secs
[-sh-4.2$ ./matrix sample1.txt
 2000 * 2000; SEQUENTIAL; 55.670000 secs
[-sh-4.2$ ./matrix sample1.txt
 2000 * 2000; SEQUENTIAL; 56.410000 secs
```

^matrix.c 10 Run Sample



```
[-sh-4.2$ ./first sample1.txt
 2000 * 2000; SEQUENTIAL; 6.450000 secs
[-sh-4.2$ ./first sample1.txt
 2000 * 2000; SEQUENTIAL; 6.240000 secs
[-sh-4.2$ ./first sample1.txt
 2000 * 2000; SEQUENTIAL; 6.350000 secs
[-sh-4.2$ ./first sample1.txt
 2000 * 2000; SEQUENTIAL; 6.310000 secs
[-sh-4.2$ ./first sample1.txt
 2000 * 2000; SEQUENTIAL; 6.280000 secs
[-sh-4.2$ ./first sample1.txt
 2000 * 2000; SEQUENTIAL; 6.360000 secs
[^[[A-sh-4.2$ ./first sample1.txt
[^[[A
 2000 * 2000; SEQUENTIAL; 6.250000 secs
 -sh-4.2$ ./first sample1.txt
 2000 * 2000; SEQUENTIAL; 6.270000 secs
[-sh-4.2$ ./first sample1.txt
 2000 * 2000; SEQUENTIAL; 6.240000 secs
[-sh-4.2$ ./first sample1.txt
 2000 * 2000; SEQUENTIAL; 6.320000 secs
 -sh-4.2$
```

^first.c 10 run sample

Part 2 Report:

In second.c my code creates a char array that is 1048576 elements long, this allows the array to be 10 megabytes in size. I populated the array with random characters. I then created two nested loops that access the memory in different intervals in size. Increasing from 128 kilobytes to just under 10 megabytes. The time is then recorded between how long it takes the memory to be accessed. To access the memory a bitshift is done so that the compiler doesn't ignore the access. Since the bitshift will always almost take the same amount of time that time can be neglected. I then had the program output these results to a CSV file named timeOut.csv. All the CSV files can be found in the tar file. I was able to estimate the size of the full cache with this method on my laptop however on the ILAB machine the results were very unusual. I suspect this is because it was running to fast to be measured the way I have implemented. I ran this code on the null.cs.rutgers.edu ILAB machine. My machine has an Kaby lake Intel Core I5 7267U processor.

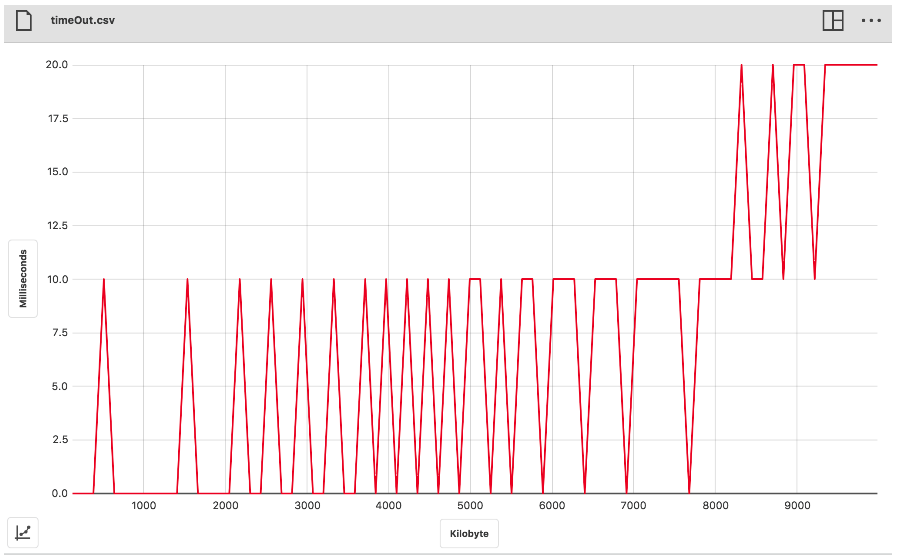
MY GUESSES:

ILAB MACHINE: null.cs.rutgers.edu

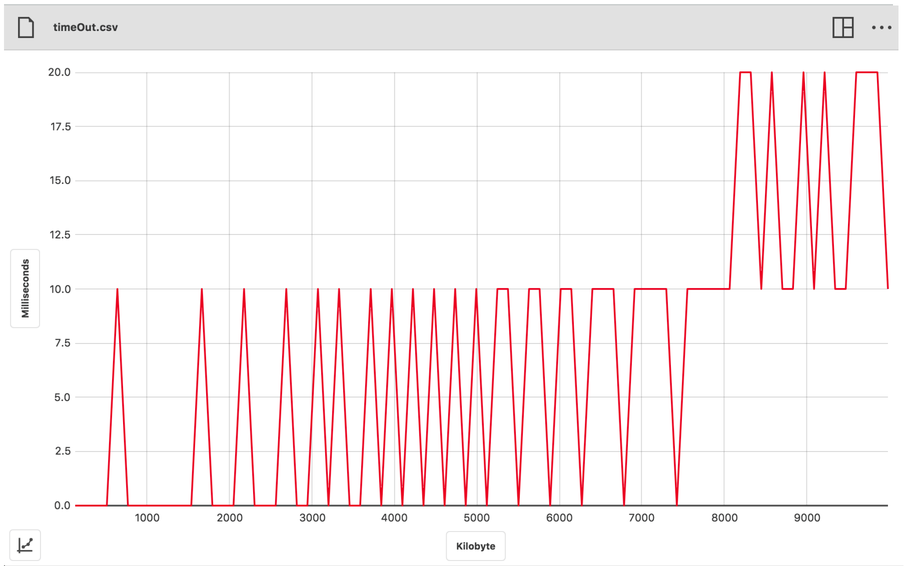
Cache Size: around 8mb (seems like it is accessing main memory after 8mb)
Set Asso: around 17 way (Number of peaks on average is around 17)

MY LAPTOP: Kaby lake Intel Core I5 7267U processor

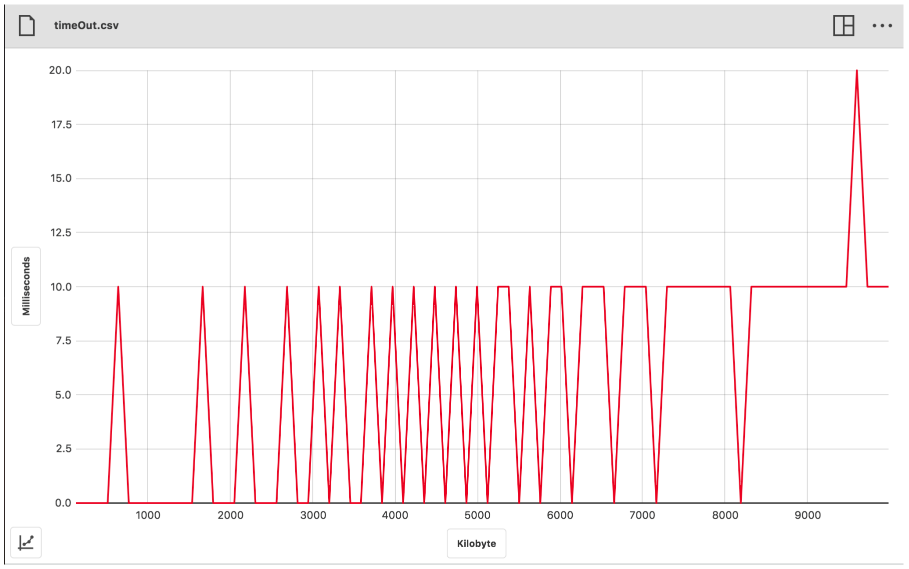
Cache size: around 4 megabytes (first peak around 500 could be L1 and the second peak around 4 mb so assuming that is the end of LLC)
Set Asso: Around 12 way (Number of peaks on average is around 12)



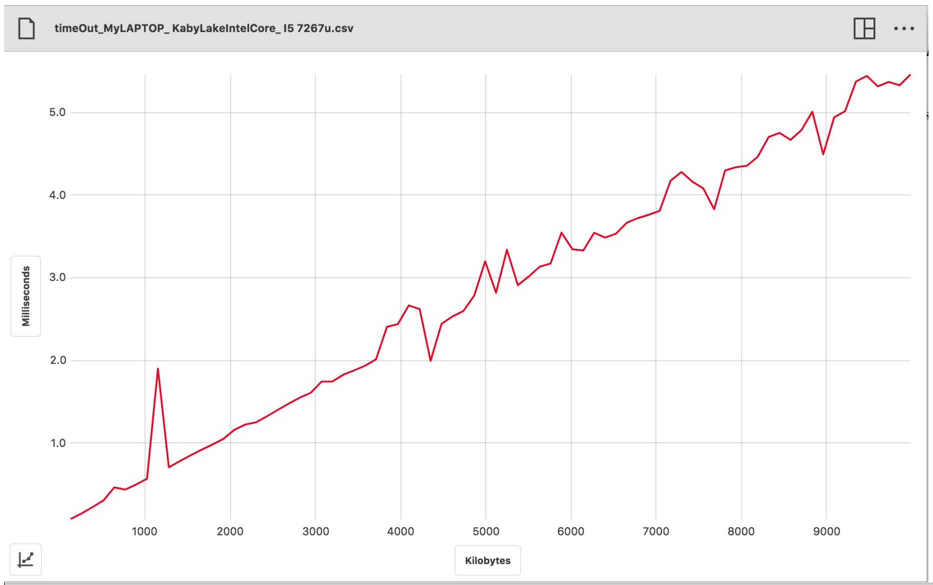
←1st Run On ILAB Machine



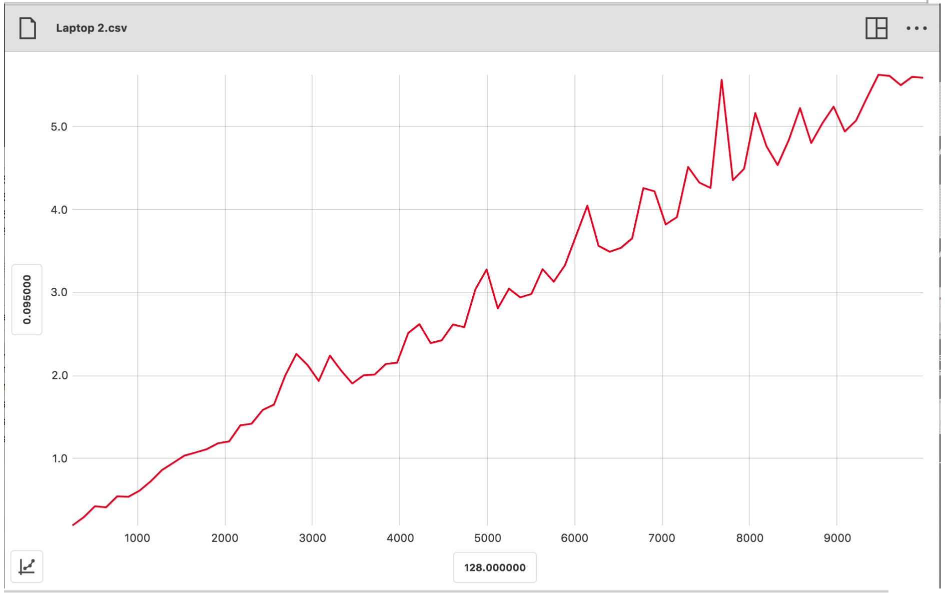
←2nd Run On ILAB Machine



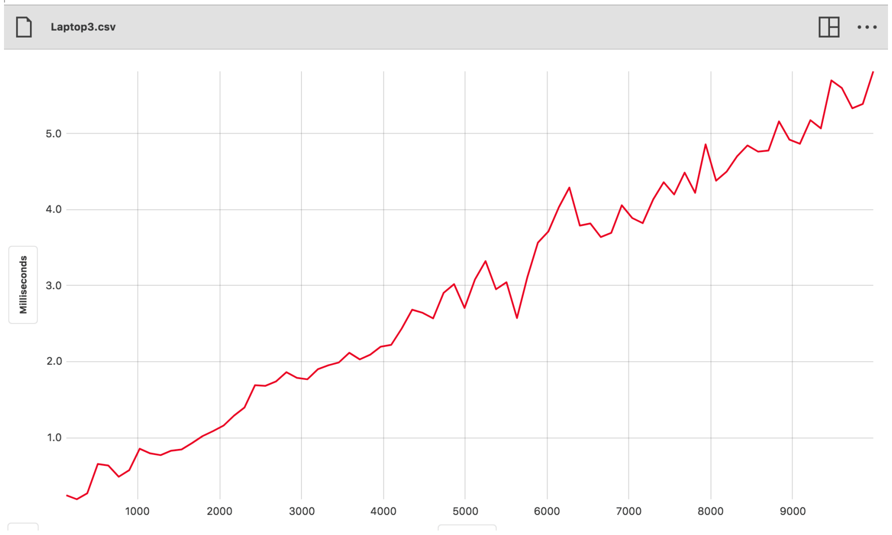
←3rdRun On ILAB Machine



←1st Run On My Machine



←2nd Run On My Machine



←3rd Run On My Machine