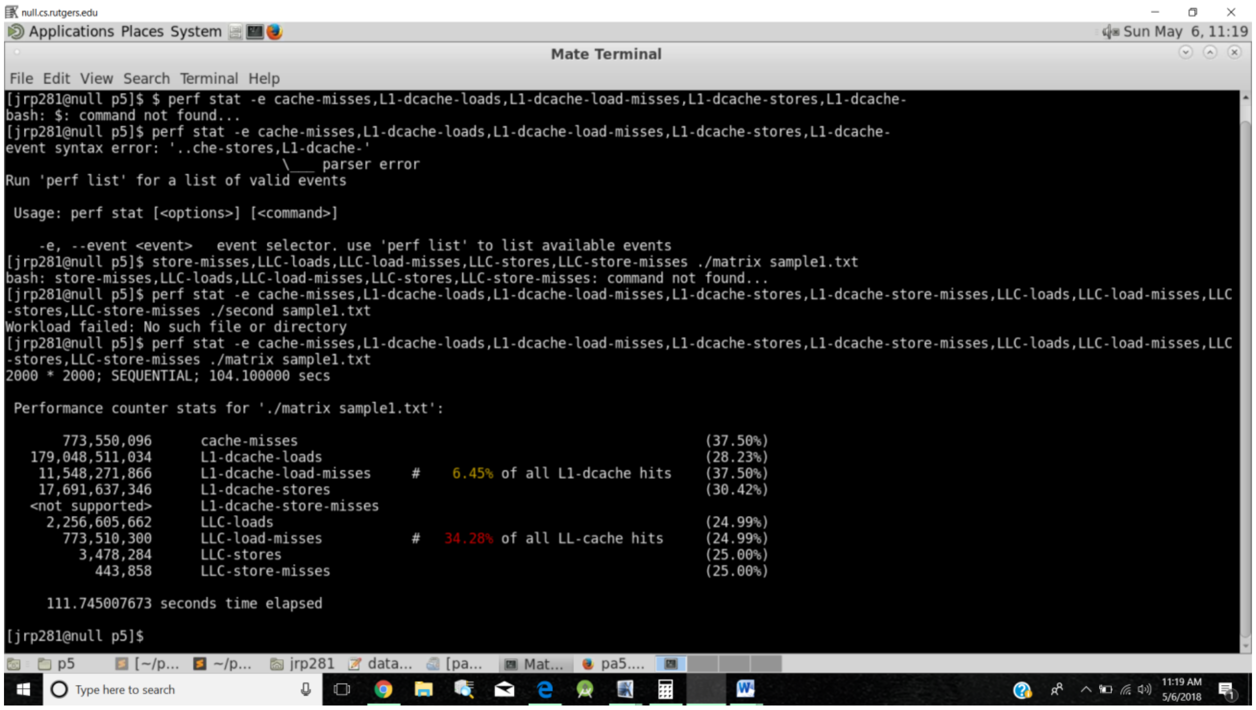
Jay Patel
Computer Architecture
Project 5
Microbench Report

Matrix Multiplication

For this assignment I had incorporated loop fusion in the multiplication part of the program. I had it allocate memory and assign variables all in one loop which caused time to increase slightly but , then Incorperated loop interchange in the program written before the actual initialization and It had decreased the time slightly.


null.cs.rutgers.edu
Applications Places System
Sun May 6, 11:19
Mate Terminal
File Edit View Search Terminal Help

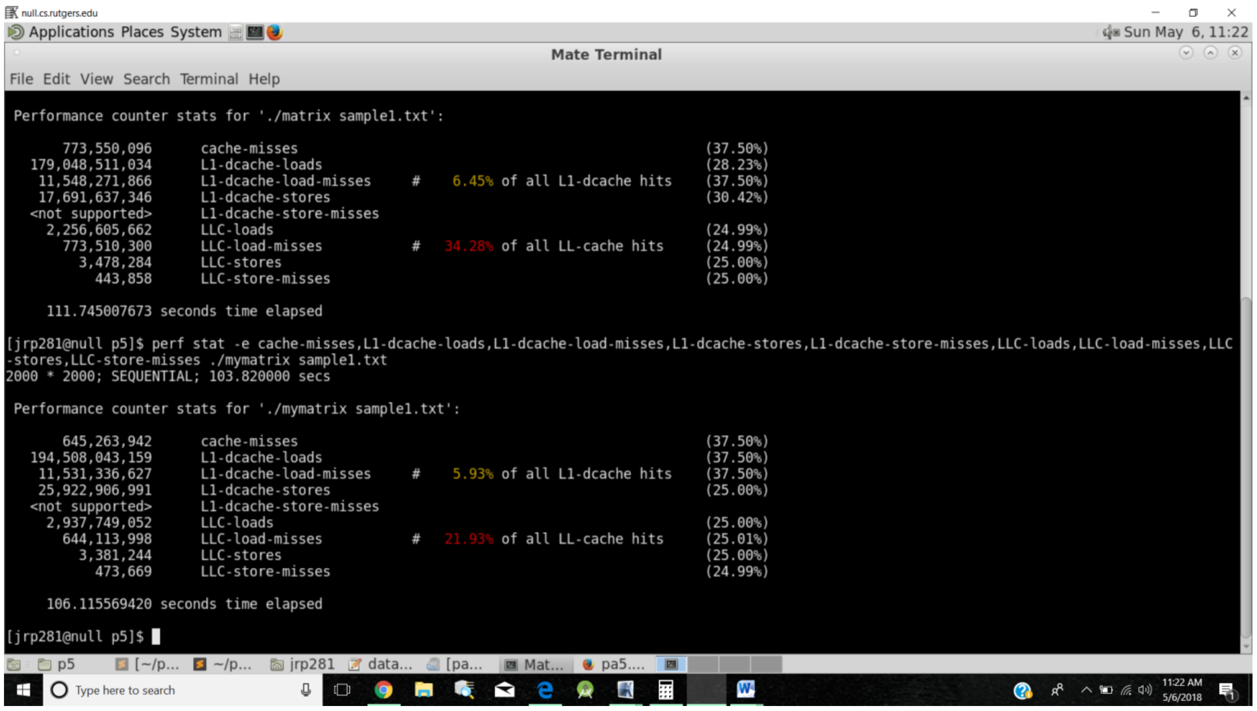
```
[jrp281@null p5]$ $ perf stat -e cache-misses,L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-dcache-
bash: $: command not found...
[jrp281@null p5]$ perf stat -e cache-misses,L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-dcache-
event syntax error: '..che-stores,L1-dcache-'
                                  \___ parser error
Run 'perf list' for a list of valid events

 Usage: perf stat [<options>] [<command>]

    -e, --event <event>   event selector. use 'perf list' to list available events
[jrp281@null p5]$ store-misses,LLC-loads,LLC-load-misses,LLC-stores,LLC-store-misses ./matrix sample1.txt
bash: store-misses,LLC-loads,LLC-load-misses,LLC-stores,LLC-store-misses: command not found...
[jrp281@null p5]$ 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 ./second sample1.txt
Workload failed: No such file or directory
[jrp281@null p5]$ 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; 104.100000 secs

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

       773,550,096      cache-misses                                                  (37.50%)
   179,048,511,034      L1-dcache-loads                                               (28.23%)
    11,548,271,866      L1-dcache-load-misses     #    6.45% of all L1-dcache hits    (37.50%)
    17,691,637,346      L1-dcache-stores                                              (30.42%)
   <not supported>      L1-dcache-store-misses
     2,256,605,662      LLC-loads                                                     (24.99%)
       773,510,300      LLC-load-misses           #   34.28% of all LL-cache hits     (24.99%)
         3,478,284      LLC-stores                                                    (25.00%)
           443,858      LLC-store-misses                                              (25.00%)

     111.745007673 seconds time elapsed

[jrp281@null p5]$
```

Jay Patel
Computer Architecture
Project 5
Microbench Report

This is the cache performance stats of the original c code that was given to us

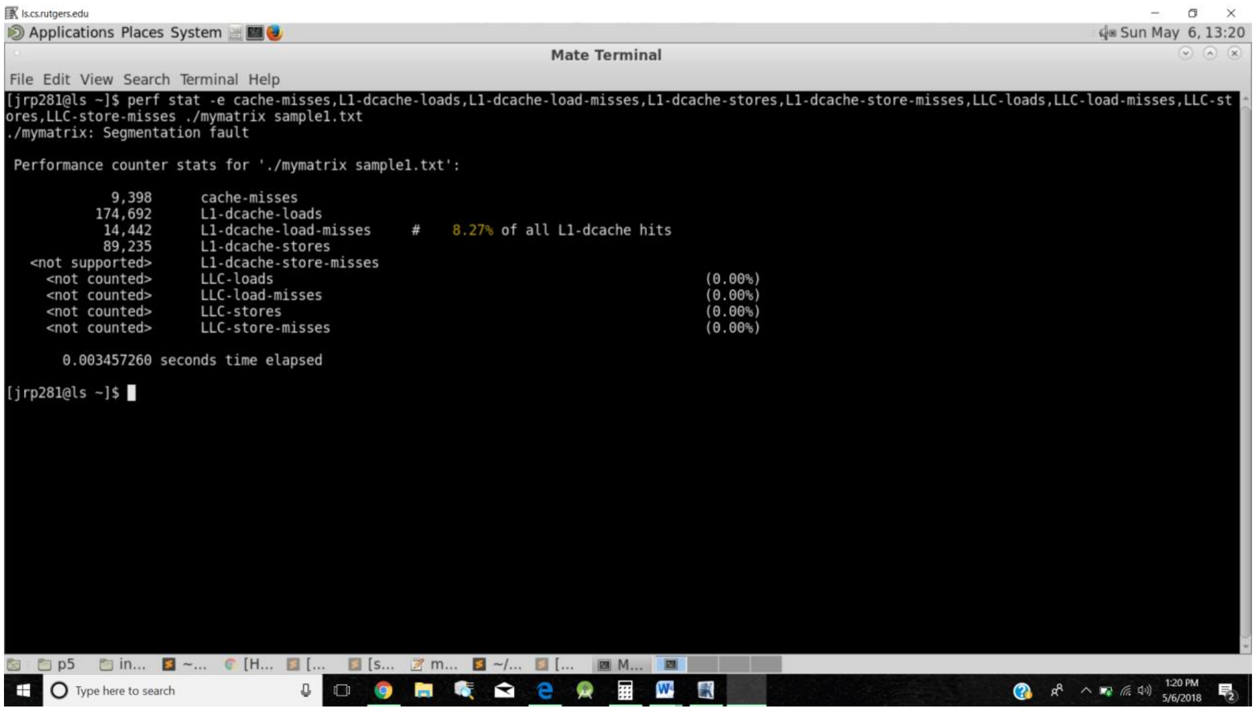
The second result is before my implementation of blocking but with just loop interchange and loop fusion have reduced the L1 cache by a little over half a percent and had taken more than 10 percent off of the LL cache.

Now that all three are implemented my results are much different



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

       773,550,096      cache-misses                                                  (37.50%)
   179,048,511,034      L1-dcache-loads                                               (28.23%)
    11,548,271,866      L1-dcache-load-misses     #    6.45% of all L1-dcache hits    (37.50%)
    17,691,637,346      L1-dcache-stores                                              (30.42%)
   <not supported>      L1-dcache-store-misses
     2,256,605,662      LLC-loads                                                     (24.99%)
       773,510,300      LLC-load-misses           #   34.28% of all LL-cache hits     (24.99%)
         3,478,284      LLC-stores                                                    (25.00%)
           443,858      LLC-store-misses                                              (25.00%)

     111.745007673 seconds time elapsed

[jrp281@null p5]$ 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 ./mymatrix sample1.txt
2000 * 2000; SEQUENTIAL; 103.820000 secs

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

       645,263,942      cache-misses                                                  (37.50%)
   194,508,043,159      L1-dcache-loads                                               (37.50%)
    11,531,336,627      L1-dcache-load-misses     #    5.93% of all L1-dcache hits    (37.50%)
    25,922,906,991      L1-dcache-stores                                              (25.00%)
   <not supported>      L1-dcache-store-misses
     2,937,749,052      LLC-loads                                                     (25.00%)
       644,113,998      LLC-load-misses           #   21.93% of all LL-cache hits     (25.01%)
         3,381,244      LLC-stores                                                    (25.00%)
           473,669      LLC-store-misses                                              (24.99%)

     106.115569420 seconds time elapsed

[jrp281@null p5]$ 
```

Jay Patel
Computer Architecture
Project 5
Microbench Report


ls.cs.rutgers.edu
Applications Places System
Sun May 6, 13:20
Mate Terminal
File Edit View Search Terminal Help

```
[jrp281@ls ~]$ 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-st
ores,LLC-store-misses ./mymatrix sample1.txt
./mymatrix: Segmentation fault

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

             9,398      cache-misses
           174,692      L1-dcache-loads
            14,442      L1-dcache-load-misses     #    8.27% of all L1-dcache hits
            89,235      L1-dcache-stores
   <not supported>      L1-dcache-store-misses
     <not counted>      LLC-loads                                                     (0.00%)
     <not counted>      LLC-load-misses                                               (0.00%)
     <not counted>      LLC-stores                                                    (0.00%)
     <not counted>      LLC-store-misses                                              (0.00%)

       0.003457260 seconds time elapsed

[jrp281@ls ~]$
```

The percentages have risen but the time has dropped dramatically to low amounts so I can conclude that blocking is the best time reducer when it comes to matrix multiplication. It allows for the cache to deal with smaller chunks of data so it is quicker to access making the time jump dramatically.

Jay Patel
Computer Architecture
Project 5
Microbench Report

(part 2)Microbench Report

This microbench report consists of three findings for 2 machines cache size, associativity and block size. The two machines that I had chosen to use are ls.cs.rutgers.edu and null.cs.rutgers.edu. my findings were that both had 32 kb caches both were 8 way set associative and both had a block size of 64 bits or 8 bytes.

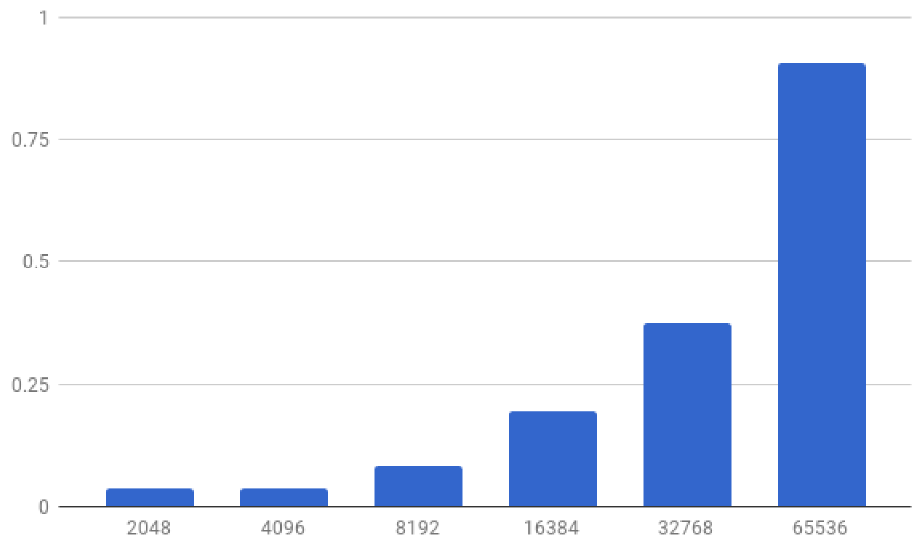
Cache size test

To find cache size I used a code written to use misses to help find the cache size by inputing a base 2 and then the code will all 2^5 to that base 2 and make It the number of bytes, when you do these you continually go up by a base 2 until you see a significant jump in timing, when you see that your program is indicating that you have hit the cache size limit. For my 258 machine I had seen this at was at 32KB due to the code being more than double what it was before, here is the output.

```
Access time for bytes of size 2048 : 0.037598 microseconds

[jrp281@null p5]$ ./bly 128

Access time for bytes of size 4096 : 0.039062 microseconds

[jrp281@null p5]$ ./bly 256

Access time for bytes of size 8192 : 0.082031 microseconds

[jrp281@null p5]$ ./bly 512

Access time for bytes of size 16384 : 0.195312 microseconds

[jrp281@null p5]$ ./bly 1024

Access time for byte of size 32768 : 0.375000 microseconds

[jrp281@null p5]$ ./bly 2048

Access time for byte of size 65536 : 0. 905632 microseconds
```

As you can see there is a large jump in the time it takes to access data so this is where the l2 cache starts. This can be seen graphically here

Jay Patel
Computer Architecture
Project 5
Microbench Report



The same was done for my 248 machine with the outputs of

Access time for block of size 8192 : 0.285156 microseconds

[jrp281@ls p5]$ ./bly 512

Access time for block of size 16384 : 0.238281 microseconds

[jrp281@ls p5]$ ./bly 1024

Access time for block of size 32768 : 1.062500 microseconds

[jrp281@ls p5]$ ./bly 2048

Access time for block of size 65536 : 0.703125 microseconds

[jrp281@ls p5]$ ./bly 4096

Access time for block of size 131072 : 1.531250 microseconds

[jrp281@ls p5]$ ./bly 8192

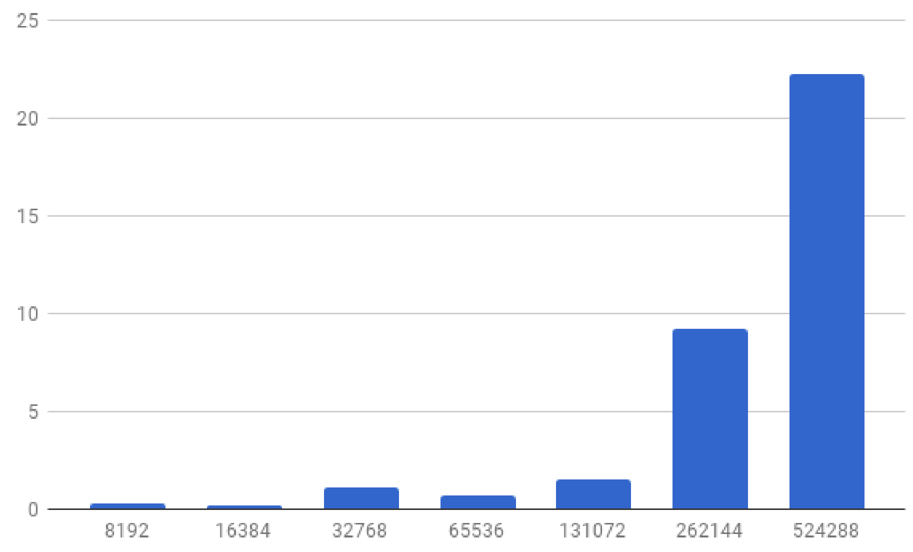
Access time for block of size 262144 : 9.187500 microseconds

[jrp281@ls p5]$ ./bly 16384

Access time for block of size 524288 : 22.250000 microseconds]

Jay Patel
Computer Architecture
Project 5
Microbench Report

As you can see there are significantly larger jumps after the 32 kb mark, this can also be seen in the visual representation



Associativity test

For the associativity test I made an array of doubles to fill the l1 cache size and looked for points that had abnormal time jumps to see where the stride to see how many ways the memory was split so what I had done was made an array of doubles that had 4096 elements because a double is 8 bytes so 8*4096 is 32 kilobytes. Then I had the program read it and output a time at each and then looked at common associativity levels 2,4,8, and 16 to see where I could see jumps. I saw minor jumps every 512 spots in the array which makes an 8 way associative cache. These following results are for the 248 machine

So at 3584 there was a jump

access time 7694 array size 3589

access time 7702 array size 3588

access time 7710 array size 3587

Access time 7718 array size 3586

access time 7726 array size 3585

access time 7733 array size 3584

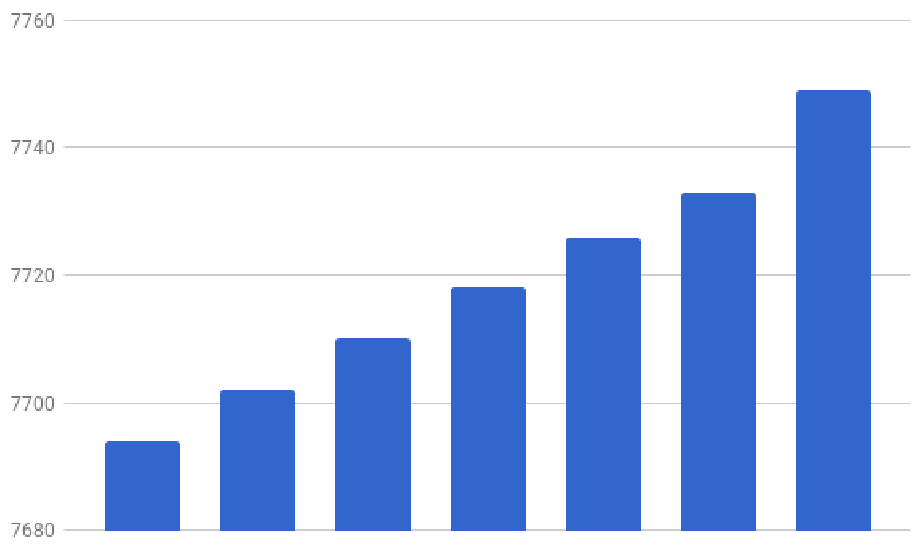
Jay Patel
Computer Architecture
Project 5
Microbench Report

access time 7749 array size 3583

the jump from 3584 to 3583 is large compared to the rest of the values

a similar jump can be seen every 512 mark

visual representation



For the 258 machine a sample output can show that the stride time goes from four to five after the 512 mark

access time 2411 array size 3586

access time 2416 array size 3585

access time 2420 array size 3584

access time 2424 array size 3583

access time 2428 array size 3582

access time 2433 array size 3581

access time 2437 array size 3580

due to it being at every 512 we can assume that the associativity is 8.

Jay Patel
Computer Architecture
Project 5
Microbench Report

Block size

Now that we have both associativity we can use mathematics to find block size. There is an equation that is cache size = assoc*(blocks per set)*block size. With my data this equation turns into 32768 = 8*(512)*blocksize so block size ends up equaling 8 bytes which makes a cache line size of 64bits.

Running my code

So for the cache size one just enter base 2 starting from 4 and keep going to notice timing differences and the associativity one has all the math in the program so just run the program without inputs. So Second.C will give you the cache size code in which you will have to run serveral times using different base 2 inputs larger than 2 and the third.c is my code to show assoc levels