CS2610 - COMPUTER ORGANIZATION AND ARCHITECTURE LAB

Performance Profiling with Perf tool

Bersilin C (CS20B013)

Introduction

The main objective of this assignment is to observe the optimization of Cache Blocking in the operation of Matrix Multiplication with the help of 2 different algorithms. One of which is a naive algorithm that causes multiple cache thrashing/ pollution occurrences. The other uses a method of Block Multiplication, which optimises the cache accesses at the data level, reducing the cache misses in the higher levels of cache in the process (eg. L1). This behaviour is studied using the perf tool available in Linux.

Method and Analysis

We begin with the naive algorithm, which follows exactly from the definition of multiplication of 2 matrices. Let the input matrices be A and B, and the product matrix be C. An element in the final matrix is the sum of products of elements in the two-parent matrix. $(c_{ij} = \sum a_{ik} * b_{kj})$. This method, when implemented, directly causes the entire B matrix to be loaded into the cache to calculate just 1 row of elements in C. If we consider a matrix size of 512*512, even a single matrix doesn't fit inside the L1-Data Cache, which has the capacity of 32 KB for a single core (Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz) (Using **getconf -a | grep CACHE,** we can get the cache sizes of all three levels of cache and in L1, both data and instruction cache separately).

Size of Cache = 32 KB = $32 * 2^{10} = 2^{15}$ Bytes

Size of 1 Matrix = $512 * 512 * 4 = 2^{19}$ Bytes

This means the entire B matrix(not precisely, as the matrix is much larger) is brought into the cache, and a significant part of it is used only once before ejecting just to be brought again. We use the Cache Blocking technique to avoid this and increase cache hits.

Here we take one particular matrix block (for example, size 64 Bytes * 64 Bytes) from the much larger matrices to calculate its product and then process them in a specific manner to get the final product matrix. As for a particular period, we are trying to find only the value of the matrix in the respective block. We avoid wasting the cache lines brought into the cache memory, improving the cache hits count. While calculating the particular block, only the strip of 64 Bytes from both the matrices is enough, a horizontal strip from matrix A and a vertical one from B.

In this method at any time, the number of usable elements present in the cache is 2*<block-size>*N, which can still be more than the L1 cache size (N = 512(max) and 2*B*N is 2^{16} (in case of block-size 64 Bytes)). But most of the cache thrashing/pollution can be avoided this way. We may get a better performance in the case of 32-Byte/16-Byte blocks as in these cases all the required elements will be present in the cache which results in very less misses.

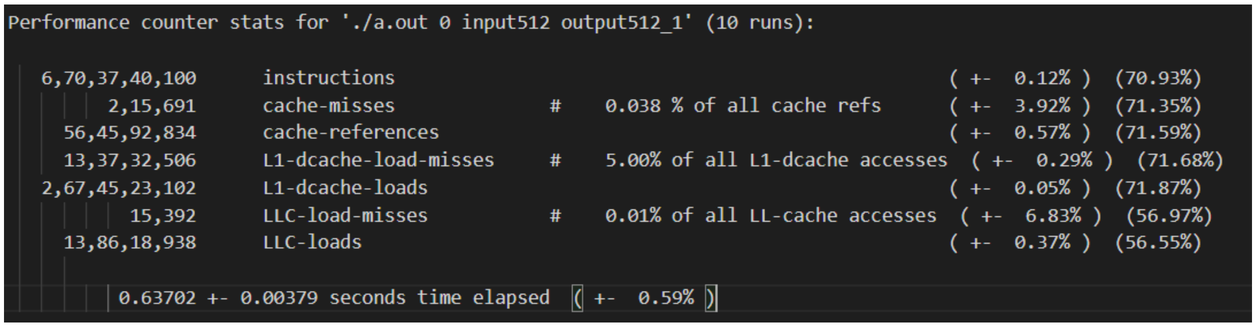
Observations

The profiling of the executable was done using the general command

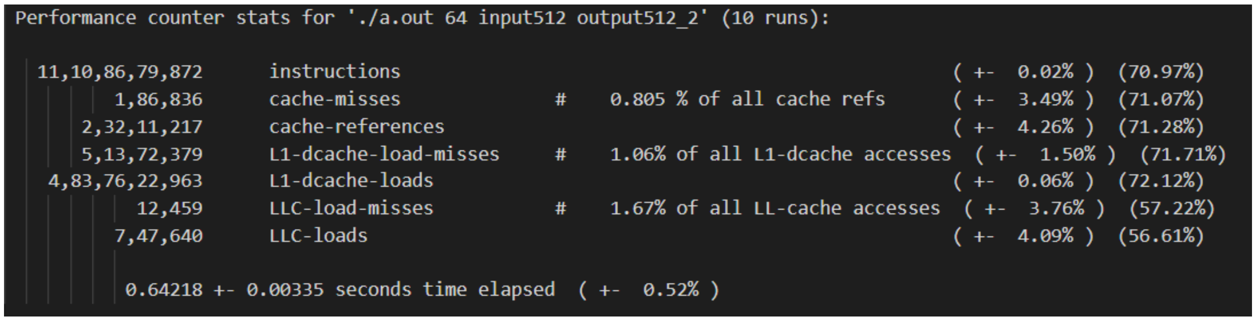
"perf stat -r <no-of-runs> -e instructions, cache-misses, cache-references, L1-dcache-load-misses, L1-dcache-loads, LLC-load-misses, LLC-loads ./a.out <block size> <inputfile> <outputfile>"

Here the block-size was varied between 0, 16, 32, 64, 128 Bytes and the various results were observed.

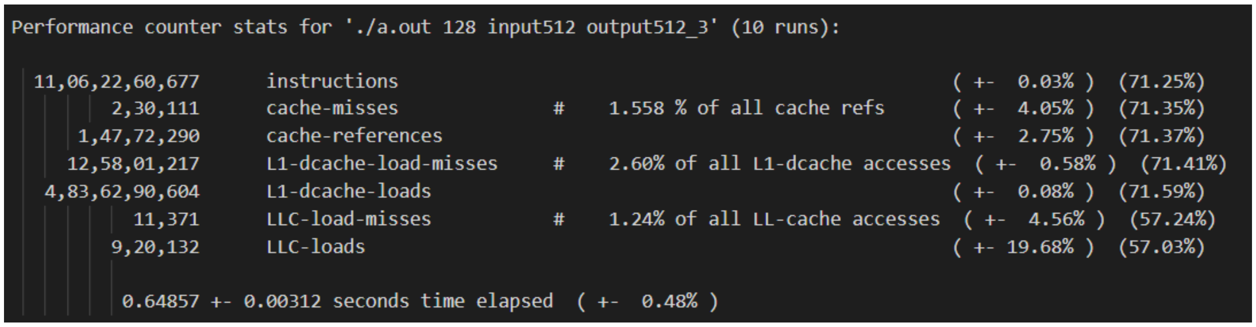
In the case of 0, 64 and 128 Bytes block-size, the data is consistent with the analysis that the block with the 64 Bytes size is better than the naive algorithm, which is technically a block of 4 Bytes. The 128 Bytes block-size case shows that the cache misses are doubled which strongly indicates the cache-line size of 64 Bytes. The data obtained for the matrix of size 512 * 512 are as follows:



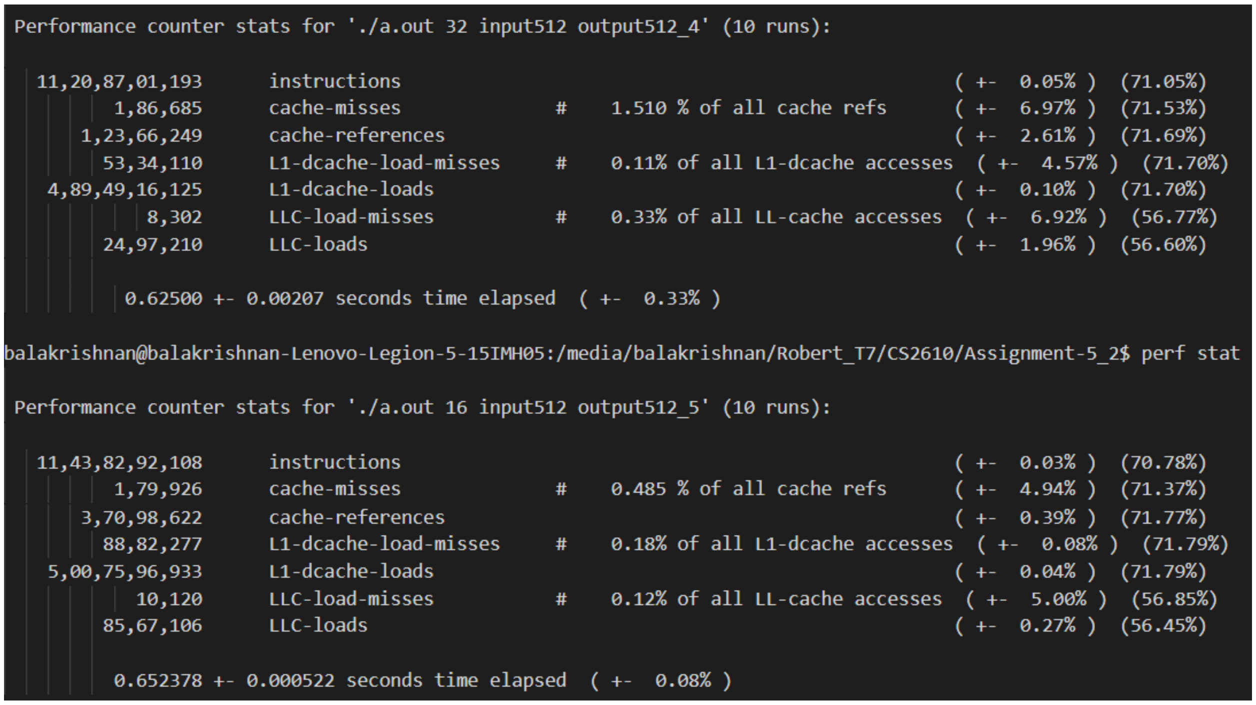
```
Performance counter stats for './a.out 0 input512 output512_1' (10 runs):

   6,70,37,40,100      instructions                                              ( +-  0.12% )  (70.93%)
        2,15,691      cache-misses              #    0.038 % of all cache refs      ( +-  3.92% )  (71.35%)
     56,45,92,834      cache-references                                          ( +-  0.57% )  (71.59%)
     13,37,32,506      L1-dcache-load-misses     #    5.00% of all L1-dcache accesses  ( +-  0.29% )  (71.68%)
   2,67,45,23,102      L1-dcache-loads                                           ( +-  0.05% )  (71.87%)
          15,392      LLC-load-misses           #    0.01% of all LL-cache accesses  ( +-  6.83% )  (56.97%)
     13,86,18,938      LLC-loads                                                 ( +-  0.37% )  (56.55%)

         0.63702 +- 0.00379 seconds time elapsed  ( +-  0.59% )
```



```
 Performance counter stats for './a.out 64 input512 output512_2' (10 runs):

  11,10,86,79,872      instructions                                              ( +-  0.02% )  (70.97%)
        1,86,836      cache-misses              #    0.805 % of all cache refs      ( +-  3.49% )  (71.07%)
     2,32,11,217      cache-references                                          ( +-  4.26% )  (71.28%)
     5,13,72,379      L1-dcache-load-misses     #    1.06% of all L1-dcache accesses  ( +-  1.50% )  (71.71%)
   4,83,76,22,963      L1-dcache-loads                                           ( +-  0.06% )  (72.12%)
          12,459      LLC-load-misses           #    1.67% of all LL-cache accesses  ( +-  3.76% )  (57.22%)
        7,47,640      LLC-loads                                                 ( +-  4.09% )  (56.61%)

         0.64218 +- 0.00335 seconds time elapsed  ( +-  0.52% )
```



```
Performance counter stats for './a.out 128 input512 output512_3' (10 runs):

  11,06,22,60,677      instructions                                              ( +-  0.03% )  (71.25%)
        2,30,111      cache-misses              #    1.558 % of all cache refs      ( +-  4.05% )  (71.35%)
     1,47,72,290      cache-references                                          ( +-  2.75% )  (71.37%)
    12,58,01,217      L1-dcache-load-misses     #    2.60% of all L1-dcache accesses  ( +-  0.58% )  (71.41%)
   4,83,62,90,604      L1-dcache-loads                                           ( +-  0.08% )  (71.59%)
          11,371      LLC-load-misses           #    1.24% of all LL-cache accesses  ( +-  4.56% )  (57.24%)
        9,20,132      LLC-loads                                                 ( +- 19.68% )  (57.03%)

         0.64857 +- 0.00312 seconds time elapsed  ( +-  0.48% )
```

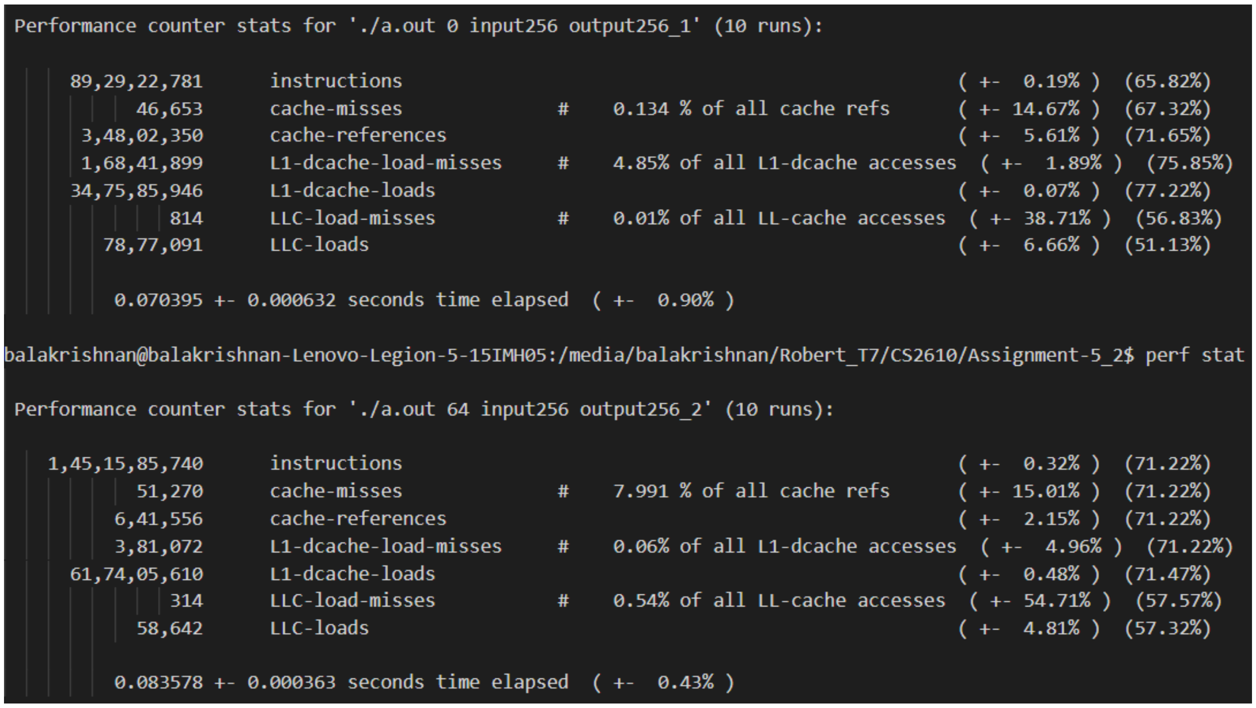


```
 Performance counter stats for './a.out 32 input512 output512_4' (10 runs):

  11,20,87,01,193      instructions                                              ( +-  0.05% )  (71.05%)
        1,86,685      cache-misses              #    1.510 % of all cache refs      ( +-  6.97% )  (71.53%)
     1,23,66,249      cache-references                                          ( +-  2.61% )  (71.69%)
       53,34,110      L1-dcache-load-misses     #    0.11% of all L1-dcache accesses  ( +-  4.57% )  (71.70%)
   4,89,49,16,125      L1-dcache-loads                                           ( +-  0.10% )  (71.70%)
           8,302      LLC-load-misses           #    0.33% of all LL-cache accesses  ( +-  6.92% )  (56.77%)
       24,97,210      LLC-loads                                                 ( +-  1.96% )  (56.60%)

         0.62500 +- 0.00207 seconds time elapsed  ( +-  0.33% )

balakrishnan@balakrishnan-Lenovo-Legion-5-15IMH05:/media/balakrishnan/Robert_T7/CS2610/Assignment-5_2$ perf stat

 Performance counter stats for './a.out 16 input512 output512_5' (10 runs):

  11,43,82,92,108      instructions                                              ( +-  0.03% )  (70.78%)
        1,79,926      cache-misses              #    0.485 % of all cache refs      ( +-  4.94% )  (71.37%)
     3,70,98,622      cache-references                                          ( +-  0.39% )  (71.77%)
       88,82,277      L1-dcache-load-misses     #    0.18% of all L1-dcache accesses  ( +-  0.08% )  (71.79%)
   5,00,75,96,933      L1-dcache-loads                                           ( +-  0.04% )  (71.79%)
          10,120      LLC-load-misses           #    0.12% of all LL-cache accesses  ( +-  5.00% )  (56.85%)
       85,67,106      LLC-loads                                                 ( +-  0.27% )  (56.45%)

        0.652378 +- 0.000522 seconds time elapsed  ( +-  0.08% )
```

As said in the analysis, we can see that optimisation works wherever the block multiplication is used for any block size. Whenever the block size is decided such that the strip of data which must be present inside the cache is less than the cache size, the efficiency is greatly increased.

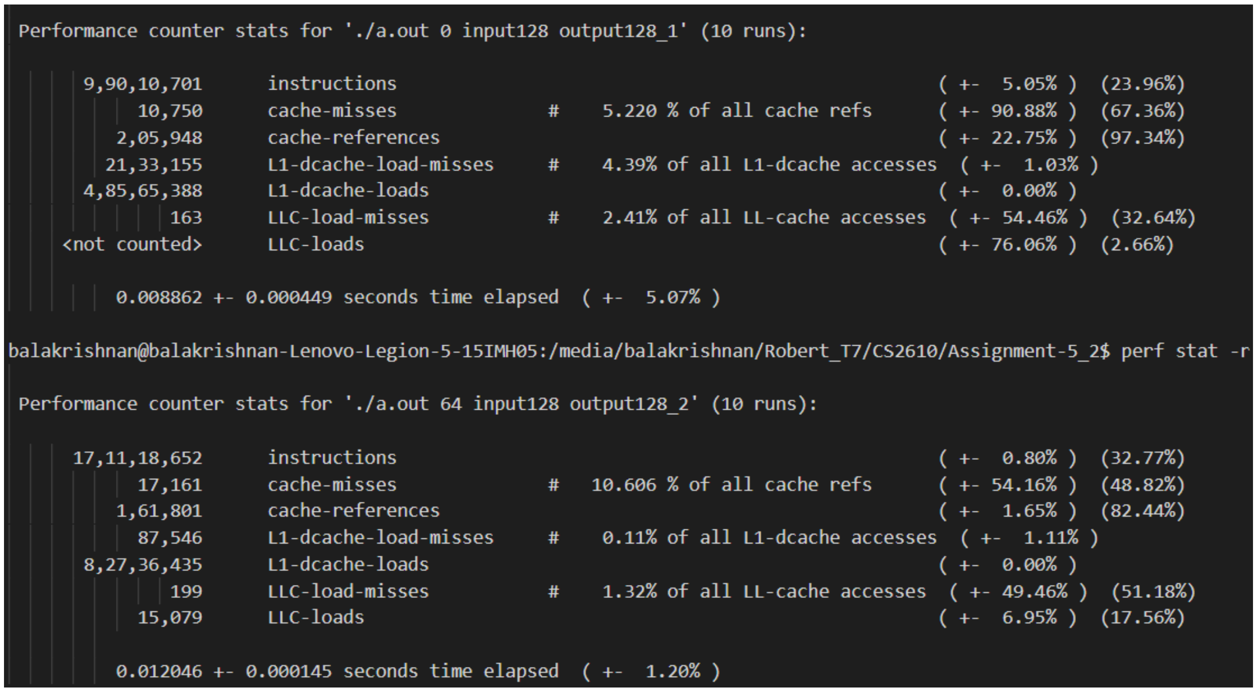
The data observed for the other matrix sizes also follow as per the above argument, the optimisation with block sizes of 64 and less and very much optimised when compared to that of the naive algorithm.

In the case of matrix sizes of 256 * 256 :



```
 Performance counter stats for './a.out 0 input256 output256_1' (10 runs):

       89,29,22,781      instructions                                                  ( +-  0.19% )  (65.82%)
             46,653      cache-misses              #    0.134 % of all cache refs      ( +- 14.67% )  (67.32%)
        3,48,02,350      cache-references                                              ( +-  5.61% )  (71.65%)
        1,68,41,899      L1-dcache-load-misses     #    4.85% of all L1-dcache accesses  ( +-  1.89% )  (75.85%)
       34,75,85,946      L1-dcache-loads                                               ( +-  0.07% )  (77.22%)
                814      LLC-load-misses           #    0.01% of all LL-cache accesses  ( +- 38.71% )  (56.83%)
          78,77,091      LLC-loads                                                     ( +-  6.66% )  (51.13%)

           0.070395 +- 0.000632 seconds time elapsed  ( +-  0.90% )

balakrishnan@balakrishnan-Lenovo-Legion-5-15IMH05:/media/balakrishnan/Robert_T7/CS2610/Assignment-5_2$ perf stat

 Performance counter stats for './a.out 64 input256 output256_2' (10 runs):

     1,45,15,85,740      instructions                                                  ( +-  0.32% )  (71.22%)
             51,270      cache-misses              #    7.991 % of all cache refs      ( +- 15.01% )  (71.22%)
           6,41,556      cache-references                                              ( +-  2.15% )  (71.22%)
           3,81,072      L1-dcache-load-misses     #    0.06% of all L1-dcache accesses  ( +-  4.96% )  (71.22%)
       61,74,05,610      L1-dcache-loads                                               ( +-  0.48% )  (71.47%)
                314      LLC-load-misses           #    0.54% of all LL-cache accesses  ( +- 54.71% )  (57.57%)
             58,642      LLC-loads                                                     ( +-  4.81% )  (57.32%)

           0.083578 +- 0.000363 seconds time elapsed  ( +-  0.43% )
```

In the case of the matrix of size 128 * 128 :



```
 Performance counter stats for './a.out 0 input128 output128_1' (10 runs):

       9,90,10,701      instructions                                                  ( +-  5.05% )  (23.96%)
            10,750      cache-misses              #    5.220 % of all cache refs      ( +- 90.88% )  (67.36%)
          2,05,948      cache-references                                              ( +- 22.75% )  (97.34%)
         21,33,155      L1-dcache-load-misses     #    4.39% of all L1-dcache accesses  ( +-  1.03% )
       4,85,65,388      L1-dcache-loads                                               ( +-  0.00% )
               163      LLC-load-misses           #    2.41% of all LL-cache accesses  ( +- 54.46% )  (32.64%)
     <not counted>      LLC-loads                                                     ( +- 76.06% )  (2.66%)

          0.008862 +- 0.000449 seconds time elapsed  ( +-  5.07% )

balakrishnan@balakrishnan-Lenovo-Legion-5-15IMH05:/media/balakrishnan/Robert_T7/CS2610/Assignment-5_2$ perf stat -r

 Performance counter stats for './a.out 64 input128 output128_2' (10 runs):

      17,11,18,652      instructions                                                  ( +-  0.80% )  (32.77%)
            17,161      cache-misses              #   10.606 % of all cache refs      ( +- 54.16% )  (48.82%)
          1,61,801      cache-references                                              ( +-  1.65% )  (82.44%)
            87,546      L1-dcache-load-misses     #    0.11% of all L1-dcache accesses  ( +-  1.11% )
       8,27,36,435      L1-dcache-loads                                               ( +-  0.00% )
               199      LLC-load-misses           #    1.32% of all LL-cache accesses  ( +- 49.46% )  (51.18%)
            15,079      LLC-loads                                                     ( +-  6.95% )  (17.56%)

          0.012046 +- 0.000145 seconds time elapsed  ( +-  1.20% )
```

Conclusion

The ideal block size cannot be generalised for all sizes of matrices. The perfect block size is when the cache can be filled efficiently to reduce cache misses, i.e., 2*<block-size>*N must be less than total-cache-size.

In our case, the ideal block size for a 512*512 matrix is 32/64 Bytes, while for the other matrix sizes of 128*128 and 265*256, block size of 64 Bytes can be used to optimise the cache dependency of the code and reduce the cache misses.