Simulation and Scientific Computing 1

Exercise 01
Performance

04 Nov 2014

Vishal Boddu
Sanjay Chamoli
Akshay kamath

The Algorithm:

It can be observed that while performing a [A]Matrix[MK]-[B]Matrix[KN] multiplication, we have a stride N access for B elements. One can think of gathering the elements of columns of B into a linear array and then perform multiplication.

The naive transpose method, in which one transposes B completely and then after the transpose operation is done we can multiply, is not very efficient as we load elements of B just to store them in an array[or Vector] and them load them again to multiply with A.

A better way could be to take only one or two columns of B depending upon its size and while storing them in a vector, while they are still in the cache multiply them with rows of A. This way would result in a very good L2 cache hit rate [as will be evident with the graphs].

-

```
if B is odd
    while storing the last column of B in a vector temp
    perform multiplications with all rows of A

store two columns[Blocking] of B one in temp, the other in
temp2
multiply contents of temp, temp2 with all rows of A in an
order
using [Loop Unrolling] and [Loop Fusion].
```

-

`naïve`: Normal Matrix-Matrix Multiplication
`Opt1`: Blocking without Loop unrolling
`Opt2`: Loop Unrolling naïve Matrix-Matrix Multiplication
`Combined`: Blocking, Unrolling, Loop Fusion

Performance Graphs:

1. Runtime

We could see that BLAS has better performance with respect to Runtime. The **`OptimalTransposeMethod`** we used has comparable results. We see that the naïve code scales very poorly with N. We see that loop unrolling over the naïve implementation doesn't give much advantage.

Runtime (s)					
Size	**Naive**	**opt1**	**opt2**	**final**	**blas**
32*32	0.0001	0.0001	0.0000	0.0001	0.0000
64*64	0.0003	0.0006	0.0002	0.0002	0.0001
128*128	0.0031	0.0038	0.0023	0.0011	0.0005
256*256	0.0281	0.0286	0.0271	0.0070	0.0031
512*512	0.2350	0.2236	0.2192	0.0477	0.0227
1024*1024	2.2484	1.7730	3.2755	0.3837	0.1723
2048*2048	70.4357	14.1483	68.4177	3.3398	1.3526

Table1:Runtime(s)

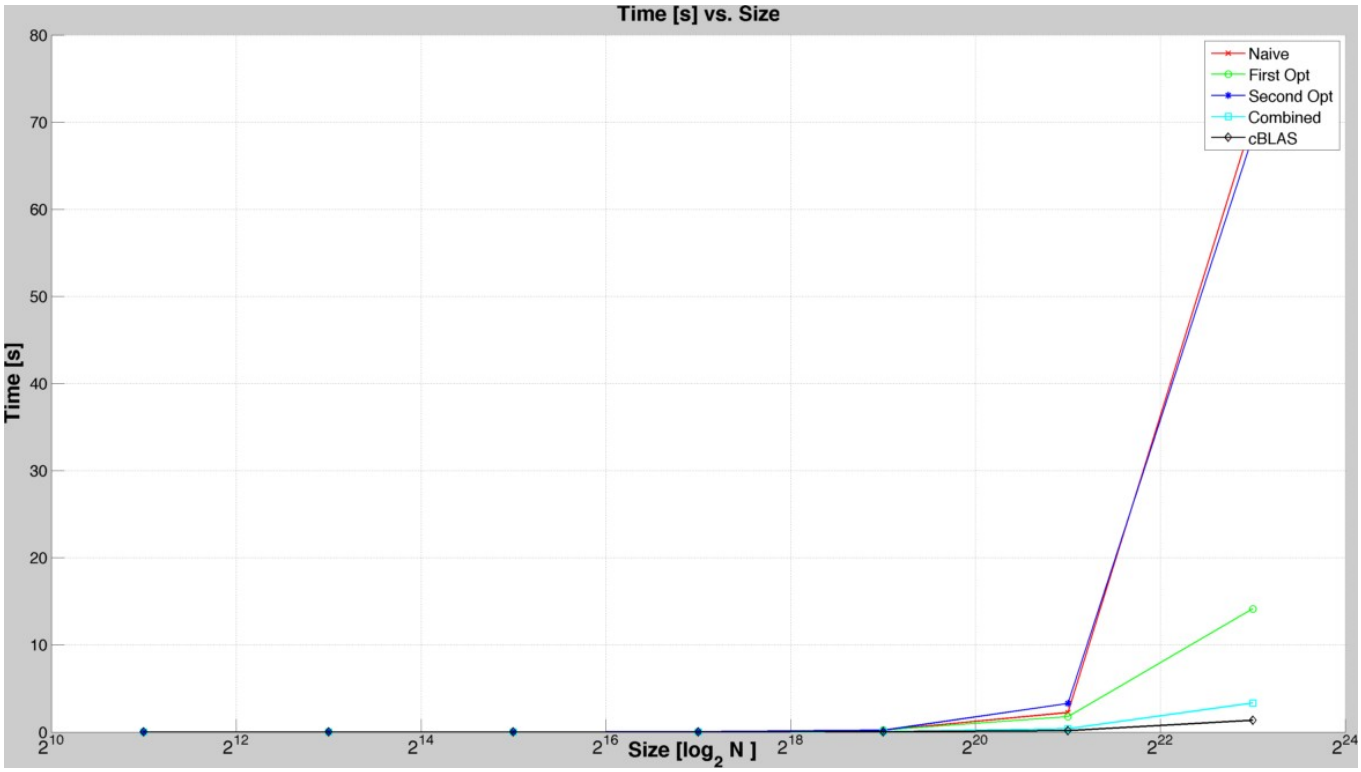

Figure 1: Shows Runtime(s)

2. L2 Cache Miss Rate

The second Opt code seems to have better L2 cache hit rate as it uses cache efficiently with blocked transpose method. The naïve implementation has poor L2 Cache hit rate. While other methods more or less have same L2 Cache Miss Rate. With such closeness in measurements, strong conclusions may not be drawn.

L2CACHE Miss Rate					
Size	**Naive**	**opt1**	**opt2**	**final**	**blas**
32*32	0.00175	0.00403	0.00218	0.00604	0.01873
64*64	0.00088	0.00109	0.00113	0.00182	0.01101
128*128	0.00812	0.00288	0.00995	0.00116	0.00577
256*256	0.11229	0.00626	0.01023	0.00441	0.00491
512*512	0.11255	0.00581	0.14909	0.00664	0.00628
1024*1024	0.11244	0.00566	0.15014	0.00959	0.00685
2048*2048	0.11433	0.00557	0.15129	0.01556	0.00709

Table 2: L2 Cache Miss Rate

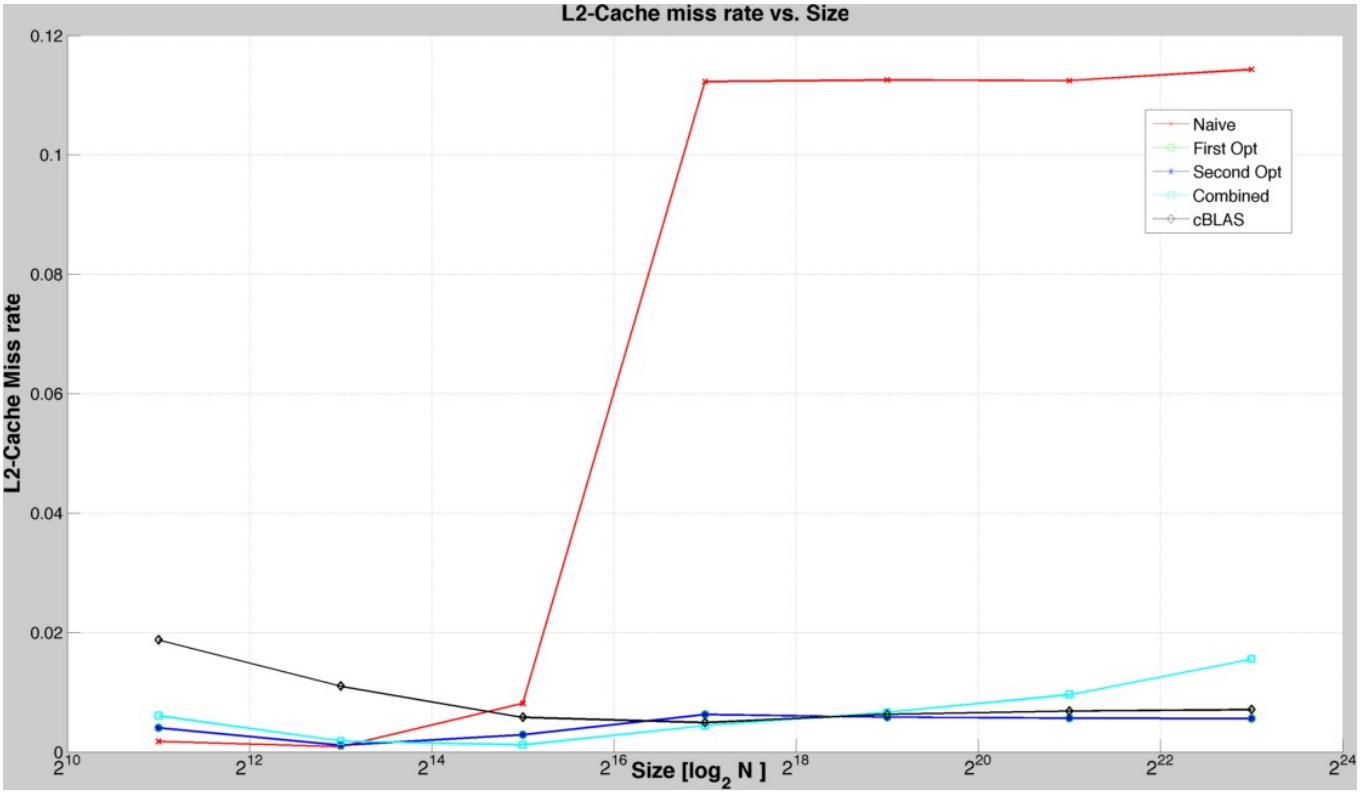

Figure 2: L2 Cache Miss Rate

3. DP FLOPS

The double precision flops are more for BLAS dgemm implementation. For all other codes flops are more or less same. It's a sign that with the data in cache, BLAS dgemm performs more operations.

DP FLOPS (MFLOPS)					
Size	**Naive**	**opt1**	**opt2**	**final**	**blas**
2048	1761.78	570.64	2538.30	16.79	911.58
8192	2003.70	957.70	2534.03	56.46	3676.31
32768	2412.47	1121.50	2510.12	141.11	7671.15
131072	2299.41	1188.30	2350.03	297.89	10459.40
524288	2029.06	1211.34	2066.28	613.19	11779.70
2097152	1101.43	1218.69	2618.99	983.64	12464.20
8388608	1239.07	1205.17	2155.32	1706.86	12710.40

Table 3: DP Flops(MFLOPS)

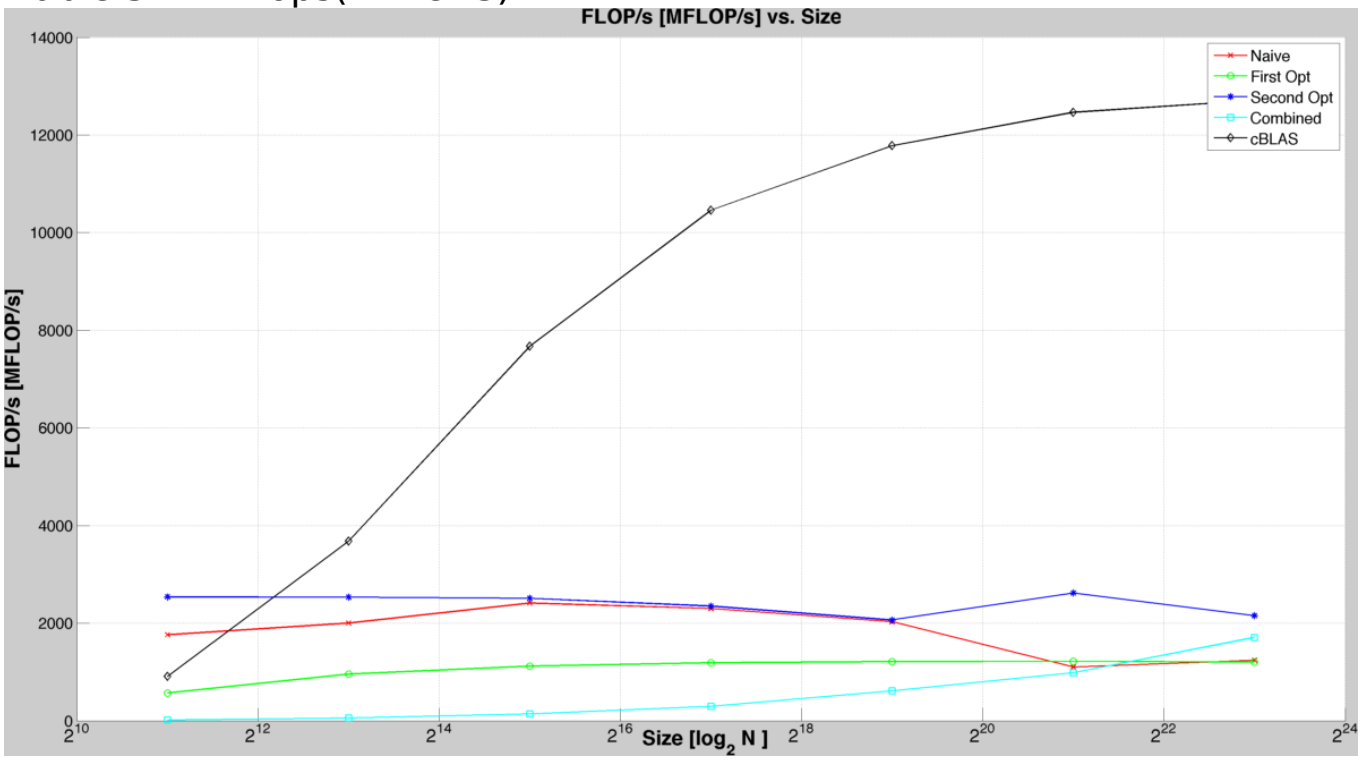

Figure 3: DP Flops

4. L2 Bandwidth

Implementations with high L2 cache hit rate don't saturate L2 bandwidth. Naive code and second Opt code have poor cache hits so its quite evident that they would have lot of traffic through L2 and L2 Bandwidth is more. For other codes L2 hit rate is quite high.

L2 Bandwidth					
Size	**Naive**	**opt1**	**opt2**	**final**	**blas**
2048	1390.59	1348.62	1933.84	132.24	2369.16
8192	16386.90	3072.85	13492.20	241.76	5191.08
32768	49983.00	2948.70	72165.20	449.98	7517.80
131072	43242.00	2712.35	73735.70	724.03	7118.13
524288	42712.50	2609.45	43215.10	1243.41	7547.37
2097152	21039.20	2659.53	10133.00	2325.21	7344.02
8388608	9677.13	7365.53	12324.00	10742.90	7165.28

Table 4: L2 Bandwidth

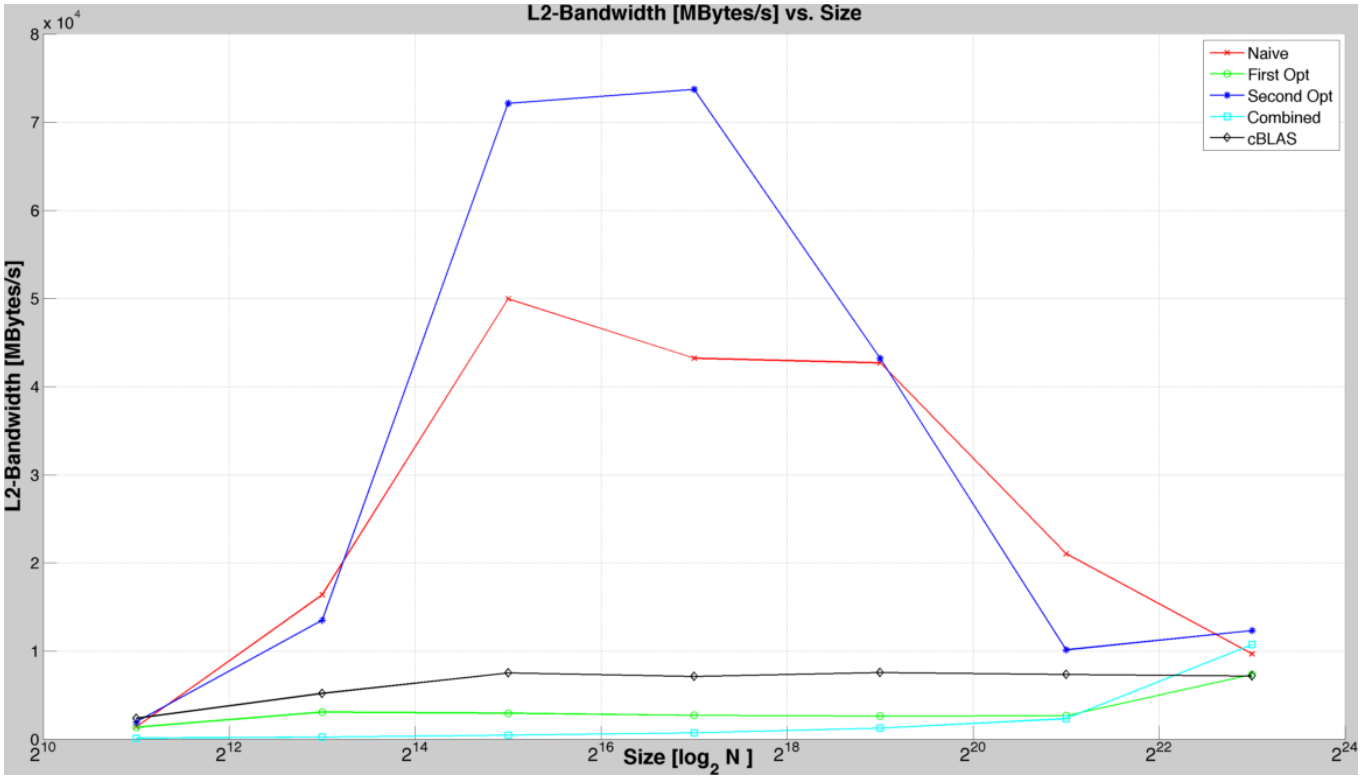

Figure 4: L2 Bandwidth