Programming Assignment #4

(The work for the optional task is in sections 3,4,5 and the code in the zip file contains the necessary changes)

1. Hardware

The CSL machine was used for compilation, execution, and benchmarking.

- Processor model: Intel® Core™ i5-11500
- Number of cores: 6
- Number of threads: 12
- 50 GB/s max memory bandwidth

2. Operating System and Compilation:

The operating system was Ubuntu 22.04.2 LTS

The intel compiler on the CSL machine was activated using the command:

```
source /s/intelcompilers-2019/bin/iccvars.sh intel64
```

This setup MKL as needed (Even when measuring the timings for the openmp implementation, the reference implementation used MKL).

Commands used for compilation and execution:

```
icc *.cpp -mkl -qopenmp -o main
./main
```

3. Code Changes

The code for the timer was changed so that the elapsed time was returned during the stop method. This was used to find the best time out of all the iterations that were executed for each of the kernels. The files changed for this were main.cpp and Timer.h

The matrix size and block size was changed from Parameter.h. The variable OMP_NUM_THREADS was set and updated to vary the thread count.

Optional Task:

To accomadate rectangular matrix sizes, 4 different matrix sizes were added to the Parameters.h file. Using these sizes, the allocations of the matrices and their casting was changed to account for the different columns/rows in matrices A,B,C. The function to initialize the values of matrices A,B was also changed. In the openmp implementation, both the matrix multiply functions were changed to accomadate for the different rows and columns. The number of blocks in these rows and columns was also recalculated. The changes to the MKL implementation and the reference function in the OpenMP implementation involved changing the matrix dimensions passed into cblas_cgemm.

4. Performance Tables

OpenMP implementation (DenseAlgebra-GEMM_Test_0_10):

Matrix Size	Threads	Block Size	Time (ms)
1024x1024	1	16x16	75.16
1024x1024	1	32x32	68.39
1024x1024	1	64x64	81.81
1024x1024	12	16x16	14.43
1024x1024	12	32x32	13.70
1024x1024	12	64x64	18.33
2048x2048	1	16x16	899.60

2048x2048	1	32x32	617.34
2048x2048	1	64x64	683.32
2048x2048	12	16x16	160.35
2048x2048	12	32x32	115.64
2048x2048	12	64x64	143.66
4096x4096	1	16x16	10068.50
4096x4096	1	32x32	5549.89
4096x4096	1	64x64	6175.58
4096x4096	12	16x16	1756.48
4096x4096	12	32x32	943.87
4096x4096	12	64x64	1246.34

MKL implementation (DenseAlgebra-GEMM_Test_0_1):

Matrix Size	Threads	Time (ms)
1024x1024	1	15.73
2048x2048	1	125.60
4096x4096	1	986.10
1024x1024	12	3.40
2048x2048	12	29.53
4096x4096	12	205.40

Optional Task:

OpenMP Implementation:

Matrix 1 Size	Matrix 2 Size	Threads	Block Size	Time (ms)
1024x2048	2048x4096	1	16x16	1181.56
1024x2048	2048x4096	1	32x32	715.42
1024x2048	2048x4096	1	64x64	951.39
1024x2048	2048x4096	12	16x16	165.55
1024x2048	2048x4096	12	32x32	127.76

1024x2048	2048x4096	12	64x64	201.35

MKL implementation:

Matrix 1 Size	Matrix 2 Size	Threads	Time (ms)
1024x2048	2048x4096	1	126.60
1024x2048	2048x4096	12	33.42

5. Comments

As expected, increasing the number of threads being used significantly improved the performance for all matrix and block sizes for both the OpenMP implementation and the MKL implementation.

For the block sizes, it seemed as though 32x32 was the sweet spot. For all matrix sizes and number of threads, increasing the block size from 16x16 to 32x32 resulted in much better performance. However, increasing it further to 64x64 resulted in slightly worse performance. Increasing the size to 32x32 allows more elements to be brought into the cache, so the multiple accesses of elements are done through the cache. The 16x16 size is not sufficiently large to take advantage of the opportunity. On the other hand, the 64x64 size might be too large to fit in the lower level caches, leading to slower memory accesses and worse performance. This explains how the changes in block size affected performance.

However, operating at a matrix size of 1024x1024, the change in block size from 16x16 to 32x32 only resulted in slightly better performance. This can be because the matrix size was too small to full take advantage of the improvements from more elements being cached since the number of blocks was much smaller than matrix sizes of 2048x2048 and 4096x4096.

The MKL implementation achieved about 4x better performance than the OpenMP implementation. Without the use of intrinsics, SIMD and more advanced optimizations, this is inline with what was expected.

Moreover, as the matrix size increased, the relative speed-up achieved from using all the threads was also greater and closer to the number of available cores. This is because there were more operations to parallelize and thus, more use was made of each of the cores.

Optional Task:

The MKL implementation was still 4x better than the OpenMP in terms of performance for the multithreaded execution. For the single thread execution it was about 6x better. This can be due to the optimizations made by MKL made a bigger relative difference during single threaded execution since OpenMP requires multithreading.

When comparing the multiplication of rectangular matrices and the square matrices of size 2048 (they have the same number of multiplications), the squares matrices performed better in both single and multithreaded execution. Since the rectangular matrices have different row and column sizes, they have a less regular memory access pattern when compared to square matrices. This leads to cache misses and worse performance.