ETH zürich

High-Performance Computing Lab for CSE **2020**

Student: Sina Klampt Discussed with: Alain Hügli, Anna Hutter

Solution for Project 1

Due date: 09.03.2020 (midnight)

In this project you will practice memory access optimization, performance-oriented programming, and OpenMP parallelizaton on Euler.

1. Explaining Memory Hierarchies *(30 Points)*

1.1. Identifying Parameters

Using the commands given on the exercise sheet, I was able to find the following values for the memory hierarchy on the computer node of the Euler cluster:

Main memory	32 GB
L3 cache	6 MB
L2 cache	256 kB
L1 cache	32 kB

1.2. Running Membench

In this exercise we were supposed to run the membench program on our local machine and on the Euler cluster.

The first plot (Fig. 1) represents the results on my local machine. I have a Intel Core i7 processor with 16 GB of Main Memory, 196 kB of L1 cache, 2 MB of L2 cache and 8 MB of L3 cache.

The second plot (Fig. 2) represents the results using Euler. I have used the Xeon Gold 6150 CPU. When we look at the time axis, we can easily see that the performance of the Euler cluster is much better than on my local machine.

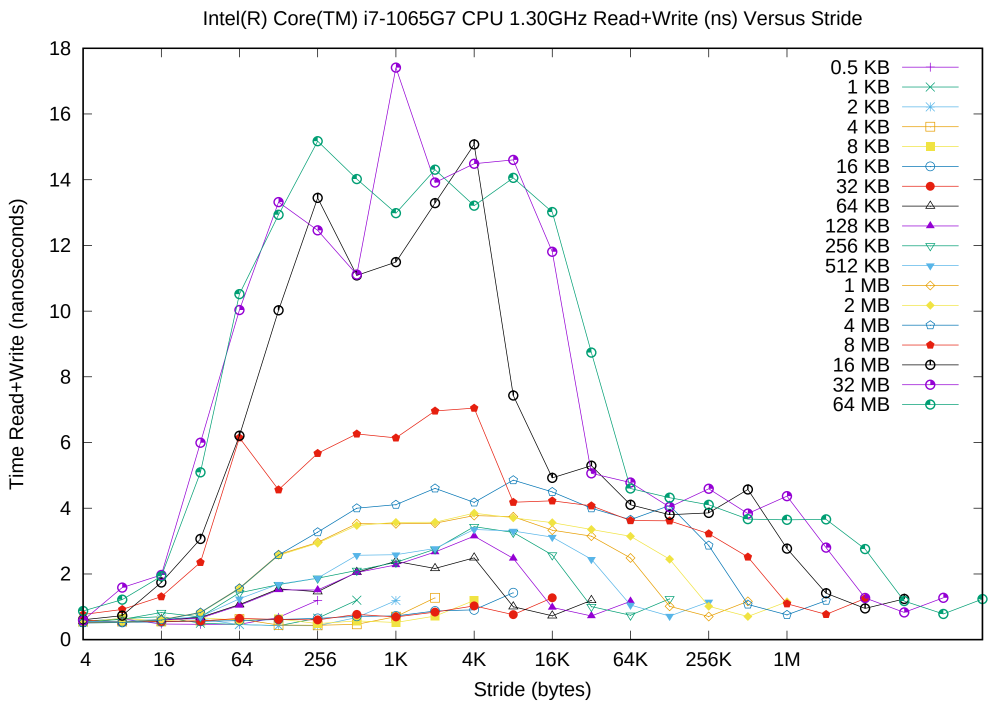

Figure 1: Plot for local machine

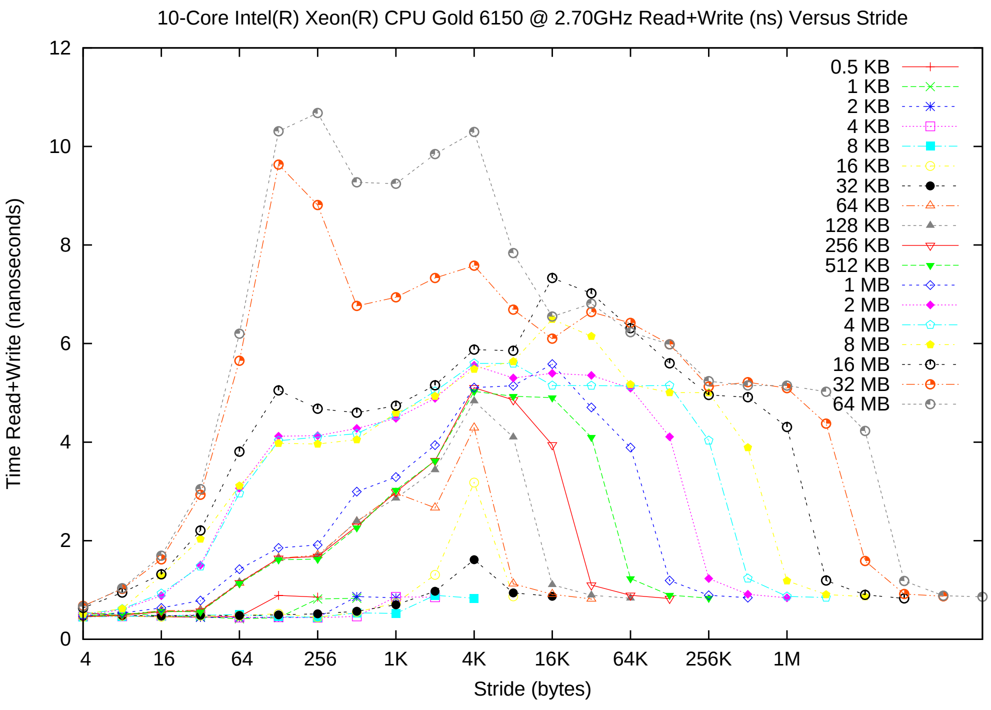

Figure 2: Plot for Euler

1.3. Memory Access Patterns

In the first case we had: csize = 128 and stride = 1. Since one integer corresponds to 4 Bytes, we have 128 times 4 which equals 512. Thus we need to look at the red line corresponding to 0.5 kB. The stride in this case is 4 Byte. In the plot we see that the read and write takes approximately 0.5 nanoseconds. Because the array fits entirely into the cache, the number of cach misses is reduced to one.

In the second case we had: csize = 2^{20} and stride = csize/2. This time we get csize = 2^{22} Byte (4 MB) and stride = 2^{21} Byte (2 MB). Thus we have to look at the light blue line with empty circles.Because we have two cache misses, it takes almost twice as long as in the previous case. We can observe that in the plot where the time is approximately 0.8 nanoseconds.

1.4. Temporal Locality

Temporal locality occurs when all elements fit into cache. This happens for large strides where we access only a few elements. The first low is because of the L2 cache and the lower times at the end are because of the L1 cache.

2. Optimize Square Matrix-Matrix Multiplication *(70 Points)*

2.1. Optimizing Part One

It took me quite some time to figure out the benefits of using the block matrix multiplication. My initial goal was to implement the proposed method and then optimizing it. However I was not able to find other ways to further improve it. I implemented the proposed method and used a block size of $\sqrt{cachesize/3}$ which led to a block size of 36 for the L1 cache. In the end I got a result slightly better than the naive method but still way worse compared to the results of the BLAS implementation. It is very impressive how much better this performs. One can see the different results in the plot below (Fig. 3).

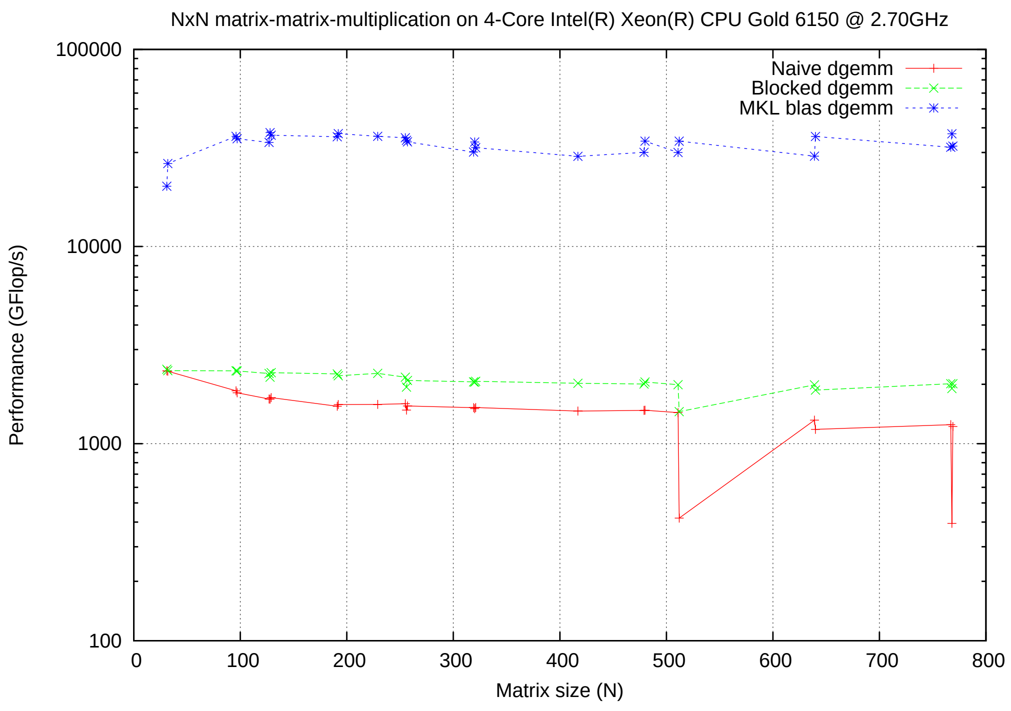

Figure 3: Plot for Blocked Matrix Multiplication

2.2. Optimizing Part Two

For the implementation in this part I used OpenMP to compute some 'for'-loops in parallel. I tried a few different options to see what performs best. In the end I used two OpenMP pragmas. To further optimize the code I initialized the conditions of the range maximum before the inner 'for'-loops. I struggled quite a lot in this exercise due to getting completely different results when compiling and running the same code over and over again. I also tried different CPUs to find the best result. For the plot below (Fig. 4) I used the 'Intel Xeon Gold 6150' CPU.

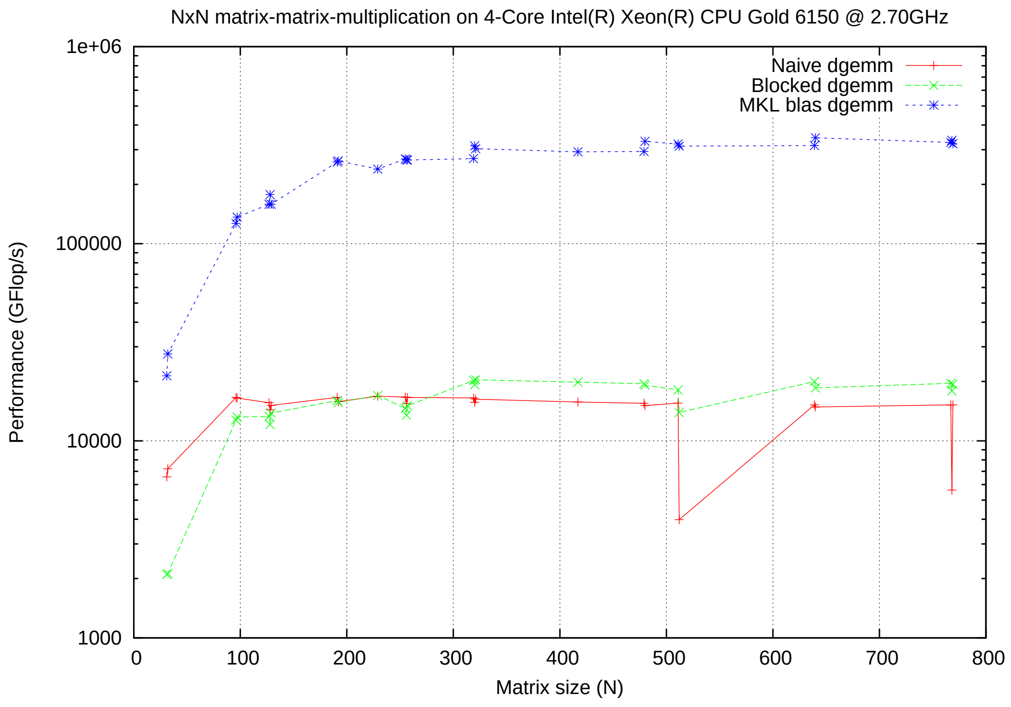

Figure 4: Plot for Parallel Implementation