Report for Lab 2

Task 1

TODO...

Task 2

a) We have applied the loop permutation technique to achieve a more locality preserving access. To be more specific, we have reordered the loops in the following way.

```
for (int row = 0; row < m1_rows; ++row)
  for (int k = 0; k < m1_columns; ++k)
    for (int col = 0; col < m2_columns; ++col)
      output[row * m2_columns + col] += ...;
```

b) Before the transformation, the total running time is 67.508 550 s, whereby the running time of the `dot` function is 64.991 598 s (96.271 654 % of total running time). After the transformation, the total running time is 10.498 188 s, whereby the running of the `dot` function is 7.994 576 s (76.151 964 % of total running time). The amount of running time that is spent in the `dot` function has been reduced significant from 96.271 654 % to 76.151 964 % (difference 20.119 69 %).

Task 3

TODO...

Task 4

a) The following snippet shows our implementation of the parallel version of the Gemm kernel.

```
...

const int chunk = m1_rows / num_partitions;
const int remainder = m1_rows % num_partitions;

pthread_t threads[num_partitions];

for (int i = 0; i < num_partitions; ++i)
{
  gemm_thread_args *args = new gemm_thread_args;

  args->m1 = &m1;
  args->m1_columns = m1_columns;
  args->m1_rows = m1_rows;
  args->m2 = &m2;
  args->m2_columns = m2_columns;
```

```
    args->output = &output;

    args->row_start = i * chunk + std::min(i, remainder);
    args->row_end = (i + 1) * chunk + std::min(i + 1, remainder);

    pthread_create(&threads[i], NULL, &dot_block, args);
  }

  for (int i = 0; i < num_partitions; ++i)
    pthread_join(threads[i], NULL);

  ...
```

b) The running times for different number of threads are given by the table below.

Number of threads	Total running time (s)
1	133.080 872
2	70.213 904
4	38.302 974
8	38.577 283
16	39.085 463
32	40.197 144
64	42.522 234

c) Since the used computer has 4 processor cores, the usage of more than 4 threads does not yield any improvement. Using more than 4 threads leads to an increasing total running time because the overhead for the thread management becomes larger and the maximum number of threads that can be executed in parallel stays constant.

Task 5

TODO...