ETH zürich

High-Performance Computing Lab for CSE **2022**

Student: Nasib Adriano Naimi Discussed with: n.a.

Solution for Project 1

Due date: 07.03.2022 (midnight)

HPC Lab for CSE 2022 — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to Moodle (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, Matlab, Julia). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

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

1. Explaining Memory Hierarchies *(30 Points)*

1. See table 1

Main memory	32 GB
L3 Cache	8 MB
L2 Cache	256 KB
L1 Cache	32,32 KB

Table 1: Memory Hierarchy Euler compute node: Intel(R) Xeon(R) CPU E3-1585L v5

2. See figures 1 and 2 for the memory benchmarks.

3. As we can see in the graphs, for both $A := (csize = 128, stride = 1)$ and $B := (csize = 2^{20}, stride = csize/2)$ the loop runs extremely fast, with A running a bit faster.
   For A : The size of the array allows it to fit entirely in the L1 cache. Although the code accesses every element of the array, the code still runs quickly since it makes use of spacial locality well.
   For B : The stride is half the size of the array. The code only accesses the first and the middle element of the array. Thus the code can execute quickly as it only has to do 2 loads and 2 cold cache misses. Spacial locality cannot be utilised, due to the array being larger than a cache line.

4. csizes that are so big, that they only access the array 2 times benefit from temporal locality, as all the accessed doubles fit into one cache line and thus there is a cache hit every time. Arrays that are small enough to fit into the L1 cache also benefit from temporal locality as the whole array is already loaded into L1.

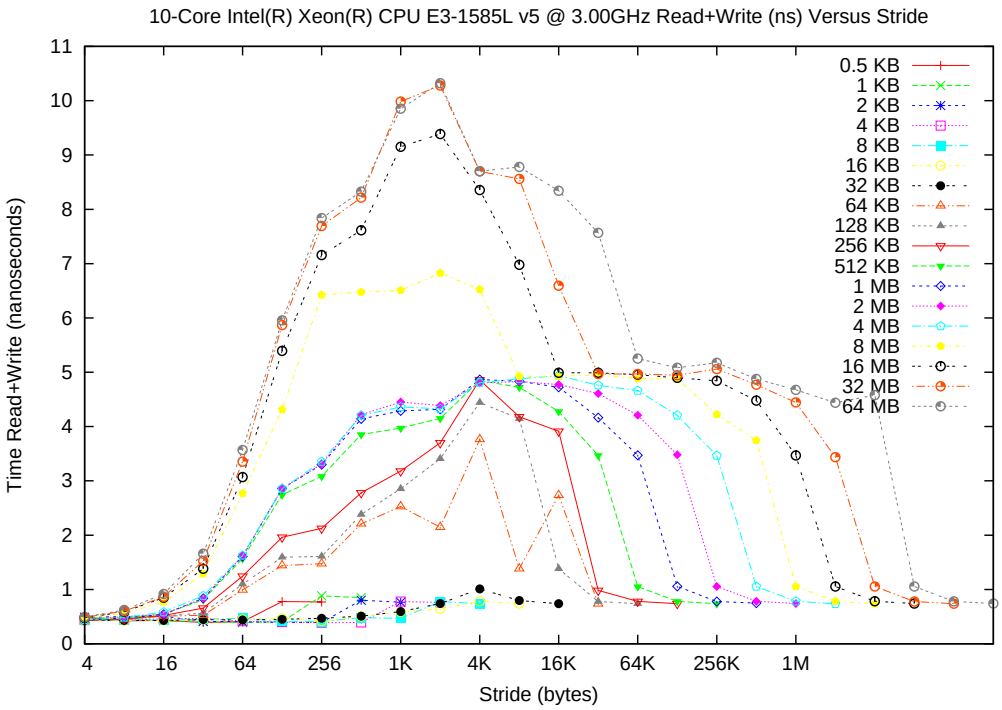

Figure 1: Memory Benchmark: Euler cluster

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

1. Here we implement the blocked version of dgemm. For the block size, we make use of the fact that the largest block size for fast memory of size M is $s = \sqrt{M/3}$ hence dividing by 8 bytes per double we get

$$\frac{\sqrt{32 * 1024/3}}{8} \approx 13 \tag{1}$$

for the size of a block in doubles. The reached speedup is about 2x.

2. for the multithread version, a directive was added for openmp inside of the two loops calling a block of the matrix. This led to approximately 10x speedup.

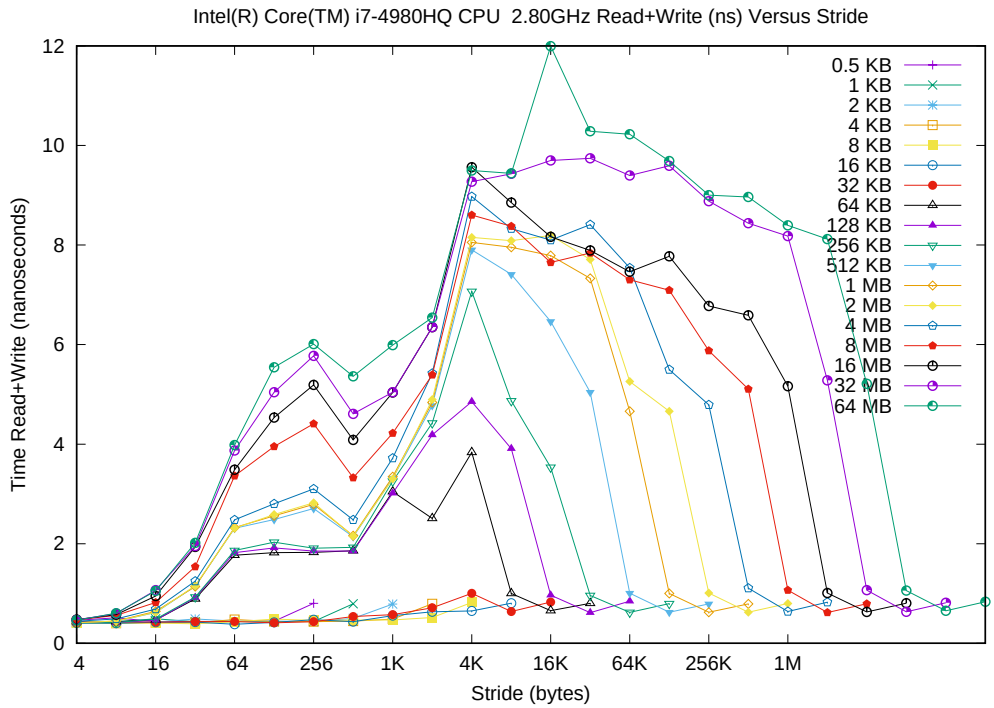

Figure 2: Local Machine: MacBook Pro 2015

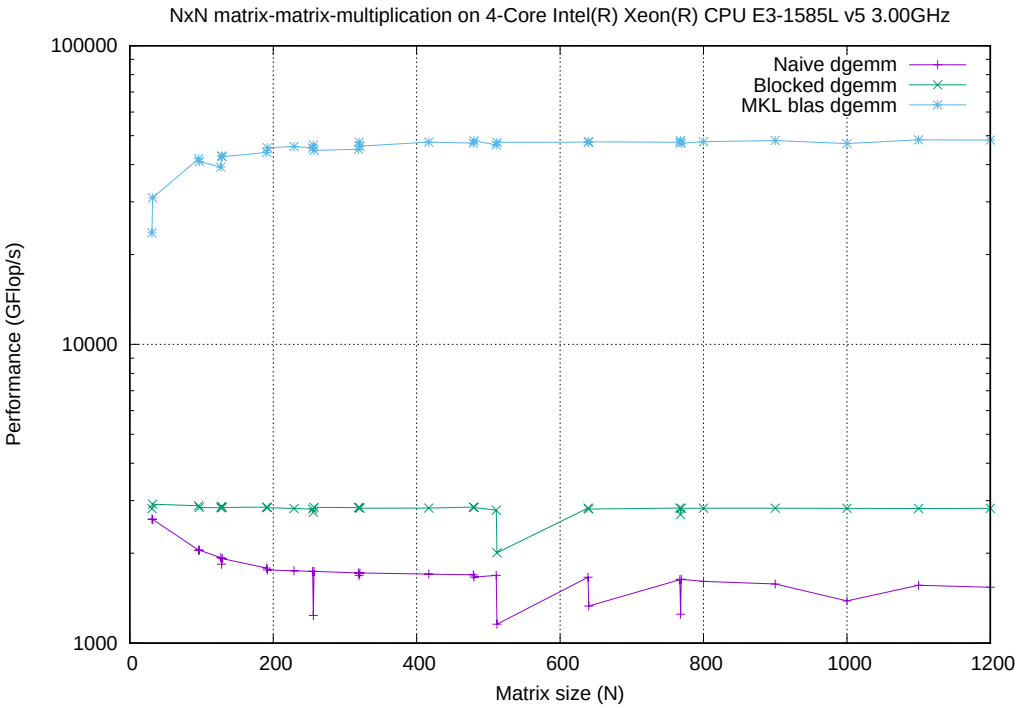

Figure 3: Performance Benchmark: Sequential on Euler cluster

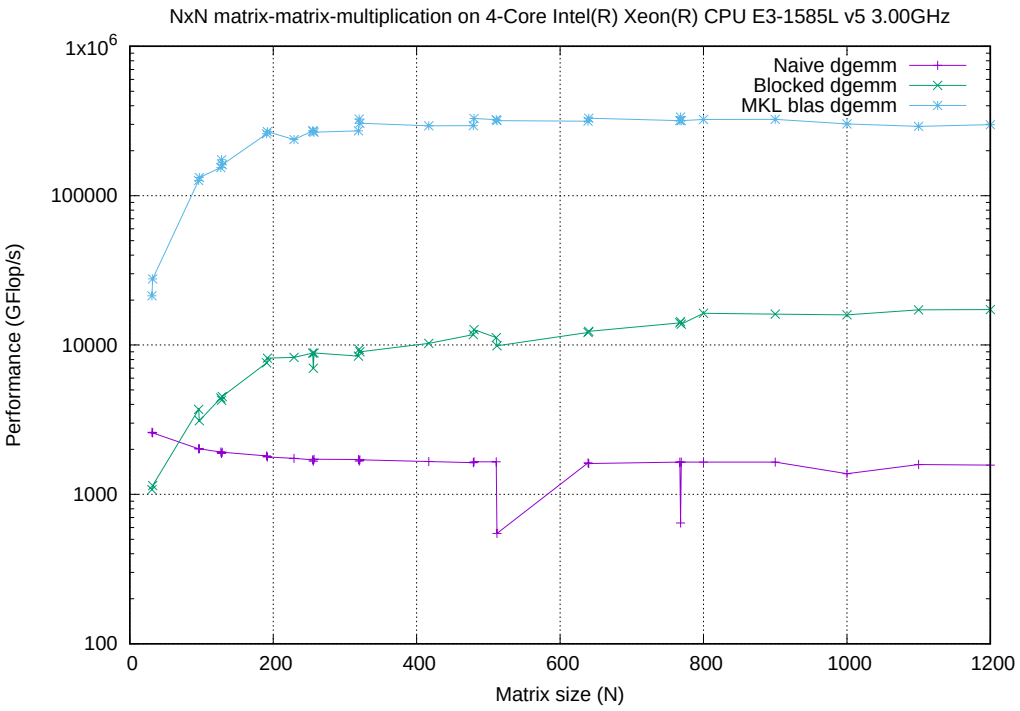

Figure 4: Performance Benchmark: Parallel on Euler cluster