Università della Svizzera italiana

Facoltà di scienze informatiche

High Performance Computing **2017**

Student: Timon Willi

Solution for Assignment 2

Due date: 24 October 2017, 13:30am

1. Explaining memory hierarchies *(30 Points)*

1.1.

Table 1. Parameters of Compute Node

Main Memory	64GB
L3 Cache	25MB
L2 Cache	256kB
L1 Cache	32kB

1.2. csize = 128 and stride = 1

With a csize of 128 B and a stride of 1, the array will be loaded completely into the L1 Cache of size 32kB. Since all the elements are available in the L1-Cache, we do not have a miss. The stride size does not matter since with any stride size, there wouldn't be a miss since all of the data is already in the L1-Cache.

1.3. csize = 2^{20} and stride = csize/2

Assuming that $2^{20} = 1'048'576$ is around 1MB, the array can be loaded completely into the L3-Cache. 256 kB are then partially loaded into the L2-Cache, from which 32kB are loaded into the L1-Cache. In the next step, since stride size is csize/2, we will have a miss in the L1-Cache and in the L2-Cache. So we need to get the rest of the array in the L3-Cache and load it into the L2-Cache, replacing what previously was in there and equivalently so in the L1-Cache. It has to do this once, since the stride is exactly have the size of the array.

1.4.

We want a small amount of Time Read + Write, so for each array size and stride there are different well-performing combinations. We see that anything up to 32KB has a very low reading and writing time, since it fits into the L1-Cache. One could argue that 32kB is the most efficient since it uses the L1-Cache the most. After 32KB there are high read and writing times, since they cannot be loaded completely into the L1-Cache. However, with increasing stride size, the loads become less and less and thus it takes less time for reading and writing. This happens because less elements need to be read and thus, there are less loadings needed.

1.5.

In Figure 1 we can see the graph of the my Laptop. As one can see, the read and writing times are considerably lower than on the cluster. One reason of this might be the architecture of the cluster and how data is being read into the working nodes. Towards a higher stride, the performances converge again, since there are less loads. This again would allow for the conclusion that the memory access is faster on the MacBook Pro than on the cluster.

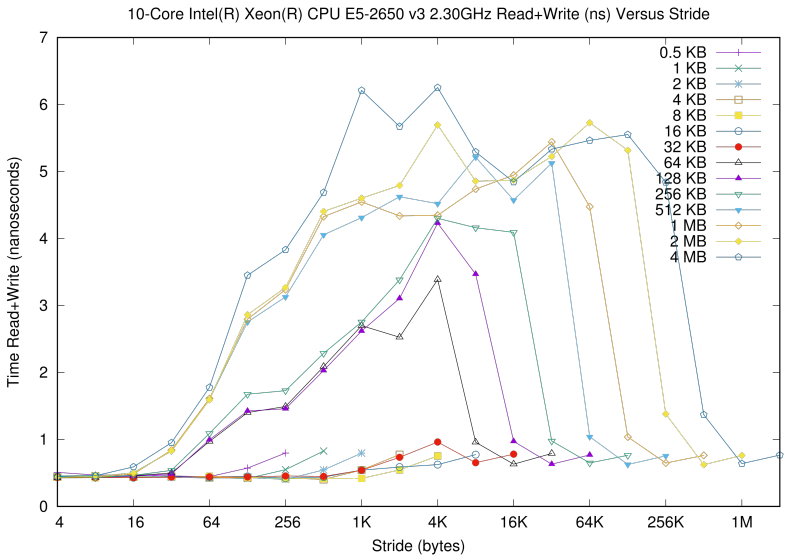

Figure 1. MacBook Pro with macOS Sierra 10.12.6, 2.8GHz Intel Core i7

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

The irregularities are caused by the fact, that the matrix size cannot be divided by the block size, leaving a residual which then has to be calculated. The green lines represents my attempt at building a more efficient matrix multiplication. The block-size that was chosen is 16. Since the arrays contain doubles, which are saved in 8 bytes, just one block will end up with 16 * 16 * 8 bytes = 2048 bytes, which could easily be loaded into the L1-Cache. In

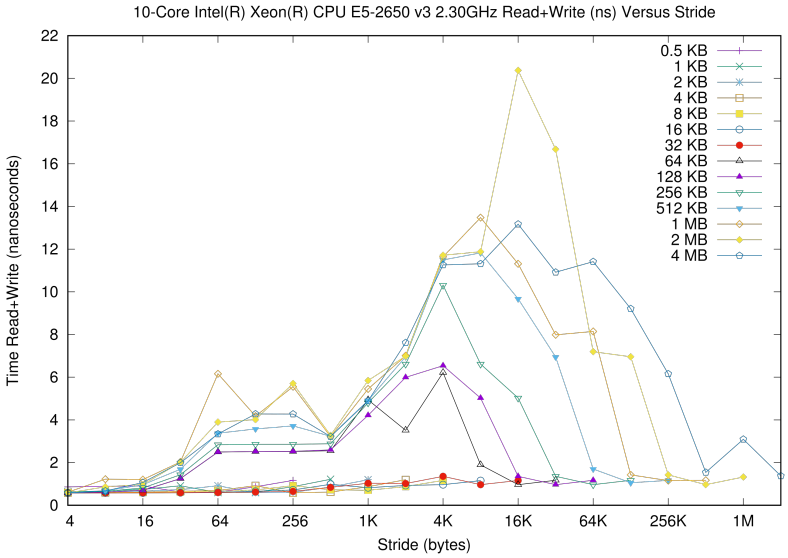

Figure 2. icsmaster

theory, a block size of 32 would also have been possible, however the performance did not change drastically. The algorithm I am using uses 5 loops. In the A-Matrix of the calculation, it selects a strip of size $1 \times block_size$ which it then multiplies with a block in the B-Matrix of size $block_size \times block_size$. The result of which is then a small strip in the C-Matrix. In the innermost loop, the small strips in A are being iterated downwards until it reaches the border of the matrix. At this point, the block in B will be moved by the block size to the right. If the block in B has reached the border on the right side of the matrix, it will then go back to the left border and go down by the block size. We thus achieve many localities. For example the strip in A is being iterated with a stride of 1. We do not need to load a strip multiple times. The block in B is being used by n strips after being loaded into memory, where n is the number of rows of A. At the end, C also has a good locality since the elements close to each other are written at a similar time. [1]. Unfortunately, the theoretical improvements do now show on the graph, showing the difficulties connected to writing my own improvements for matrix multiplication.

[1] http://csapp.cs.cmu.edu/2e/waside/waside-blocking.pdf

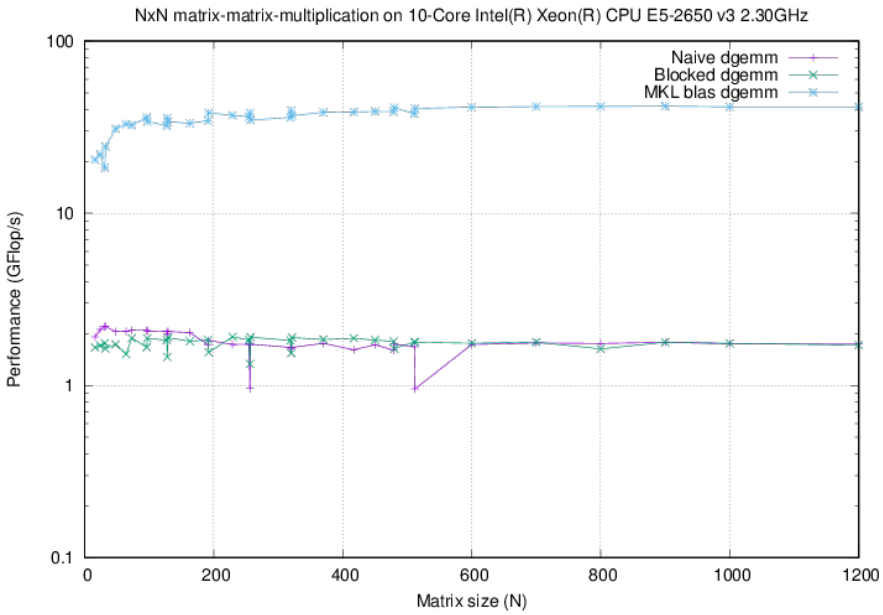

Figure 3. Block Size of 16