ParProg Testat 3

Preamble

We, Thomas Rüegg and Patrick Wissiak, are working together on this testat.

Task 1

Exercise 1

Measure the performance varying the tile size. Report your measurements and the best tile size. Is there a relationship between the tile size and the speedup? Explain.

Yes, there is a relationship between the tile size and the speedup. It depends on the GPU architecture, the available shared memory, and other hardware characteristics. It's important to experiment with different tile sizes to find the best configuration for a given problem (matrix multiplication in our case) and hardware setup.

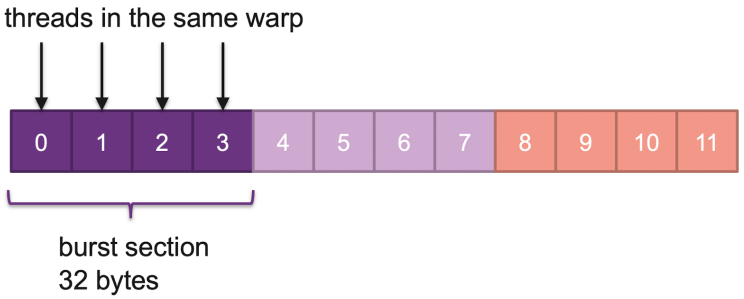
In the subsequent paragraphs, we will explain how tile size affects performance.

Shared memory usage

Shared memory is limited and much faster than global memory. By partitioning the data into tiles that fit into the shared memory, we can minimize the amount of global memory access and make more efficient use of the shared memory. Larger tile sizes can lead to better usage of shared memory, but only up to a certain point. If the tile size becomes too large, it might not fit into the shared memory, causing performance degradation.

Memory coalescing

When threads in a CUDA kernel access global memory, the memory subsystem groups these accesses into "memory bursts", if the access pattern is contiguous:



Choosing a tile size that promotes coalesced memory access will lead to better bandwidth utilization and therefore improved performance.

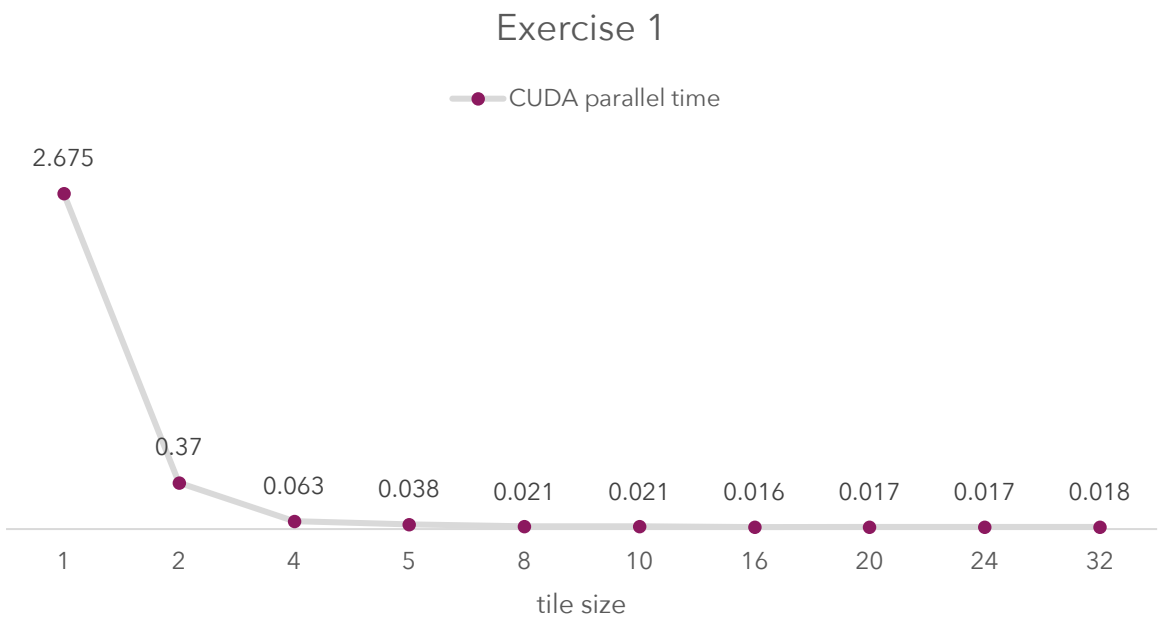
Occupancy

Occupancy is a measure of how well the GPU's computational resources are utilized. It is defined as the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Larger tile sizes typically result in more threads per block, which can increase occupancy and improve performance. However, if the tile size is too large, it may limit the number of concurrent thread blocks that can be executed per multiprocessor, reducing occupancy and performance.

Cache utilization

The choice of tile size also affects the cache utilization on the GPU. A suitable tile size can improve cache locality, which reduces cache misses and improves overall performance.

Measurements



tile size	CUDA parallel time	speedup
1	2.675	
2	0.370	7.230
4	0.063	5.873
5	0.038	1.658
8	0.021	1.810
10	0.021	1.000
16	0.016	1.313
20	0.017	0.941
24	0.017	1.000
32	0.018	0.944

It seems that a tile size of 16 will utilize the shared memory in the most optimal way.

Tile sizes above 32 will throw a `Result verification failed at element 0` error. The reason for this is that the shared memory can only fit up to 49152 bytes. The Asub and Bsub matrices, both using TILE_SIZE * TILE_SIZE * sizeof(float), will, next to some other data, demand more than the shared memory can provide.

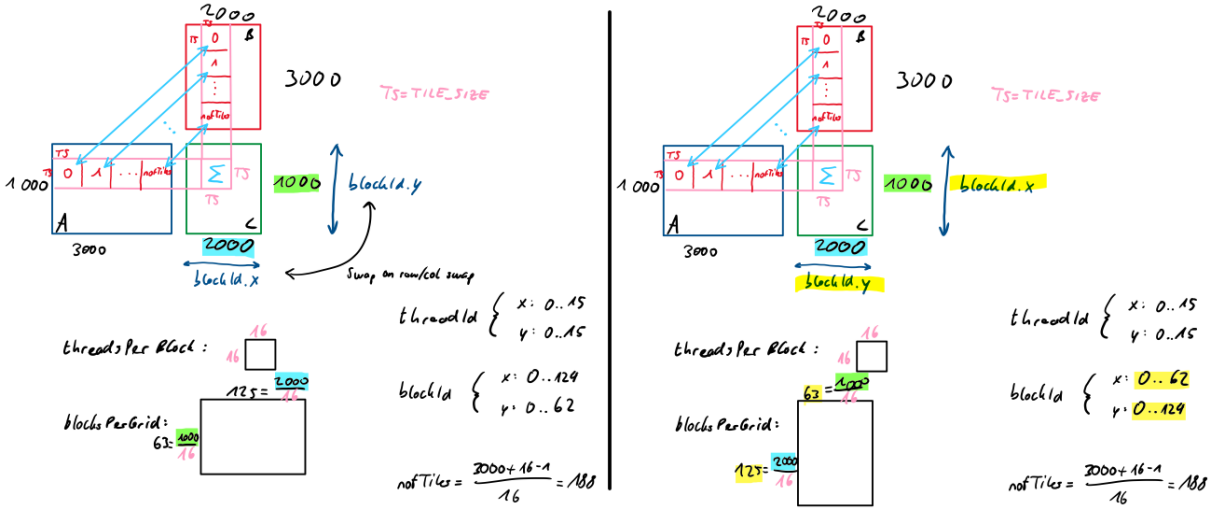
Exercise 2

Now Swap the dimensions row and column.

*int row = blockIdx.x * TILE_SIZE + tx;*
*int col = blockIdx.y * TILE_SIZE + ty;*

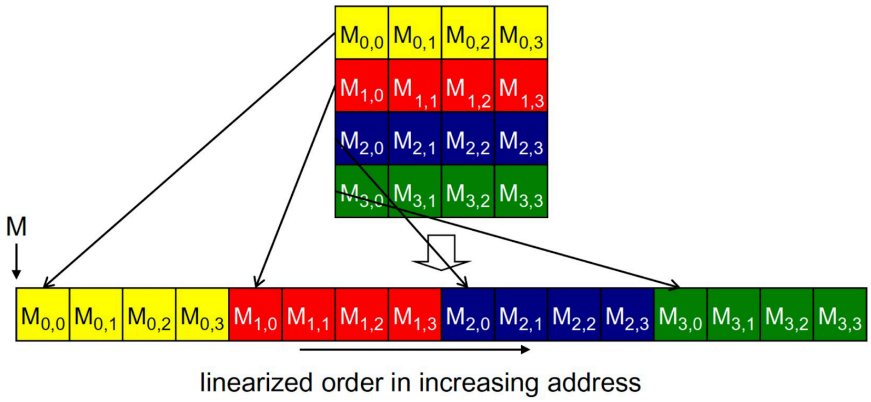
And compare the run time. Vary the tile size. Please report the experiments you performed, your measurements, speedup, plots, and the best size of the tiles for this case.

The following drawing highlights the change of swapping rows and columns:

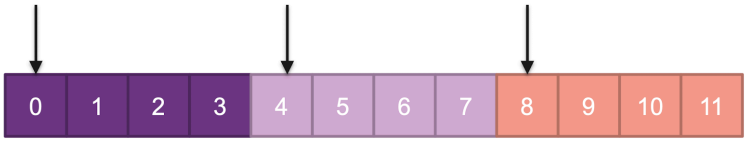


Essentially, the rows and columns of the grid swap dimensions.

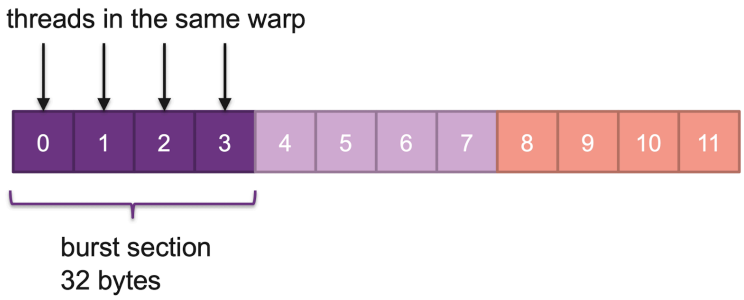
Our matrices are laid out in linearized order like the following figure presents:



At the moment, we are accessing the memory like this:

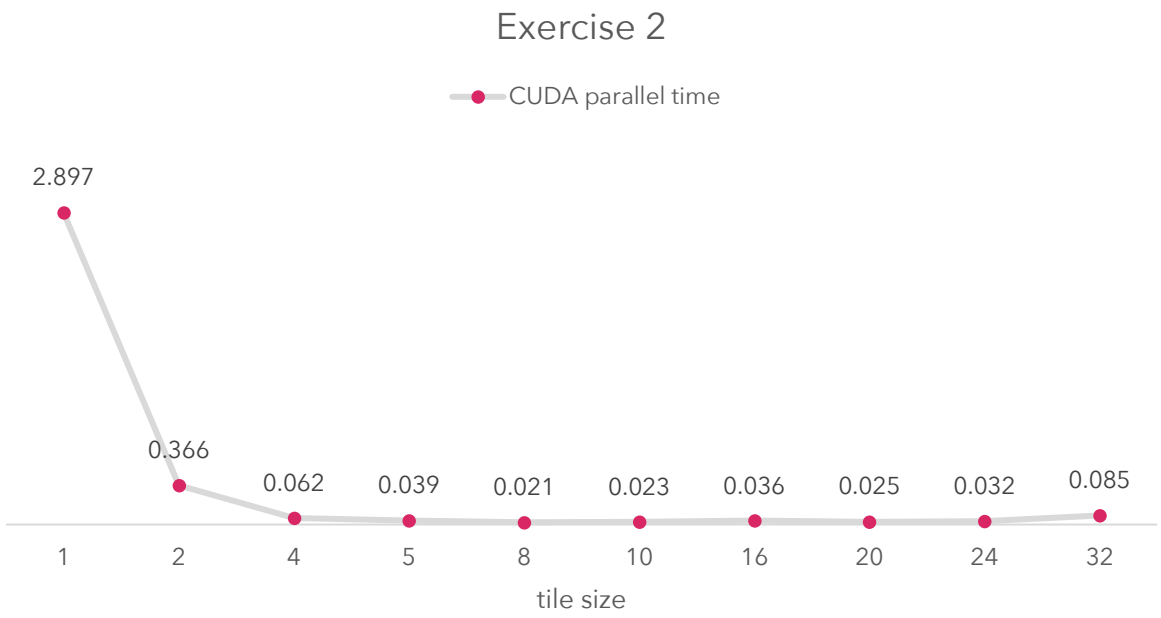


By swapping row and col, we are trying to get closer to accessing the memory in a more optimized manner, thus making use of memory bursts:



Measurements

Running the adjusted code spit out these results:

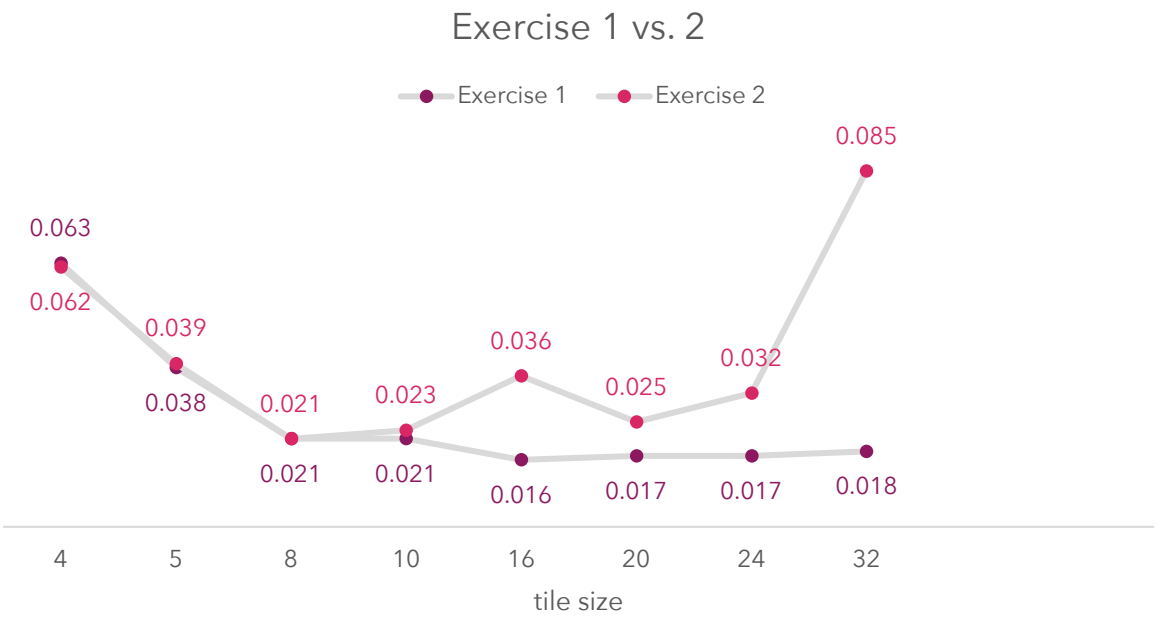


tile size	CUDA parallel time	speedup
1	2.897	
2	0.366	7.915
4	0.062	5.903
5	0.039	1.590
8	0.021	1.857
10	0.023	0.913
16	0.036	0.639
20	0.025	1.440
24	0.032	0.781
32	0.085	0.376

This time, a tile size of 8 would be preferred, as that only took 0.021 seconds. This seems to launch the optimal number of threads for the given problem.

Comparing Exercises 1 and 2

It seems as if the calculations take longer by swapping col and row, which is opposed to our expectations. The code of exercise 1 runs a bit faster than the variant of exercise 2.



The graph's x-axis is limited and only shows tile sizes from value 4. The course is the same with both graphs. However, a tile size of 16 shows a peak in the variant of exercise 2.

The code of exercise 1 runs a bit faster than the variant of exercise 2, i.e. the calculations take longer by swapping col and row, which is opposed to our expectations.

The speed of the calculations of exercise 2 might be improved further by varying matrix and block sizes.

Thanks for reading!

Best, Patrick and Thomas :)