Highly Parallel Programming of GPUs
Lab 4

Square Matrix Multiplication

A matrix multiplication of an $n \; x \; m$ matrix A and $m \; x \; p$ matrix B is defined by:

$$A \cdot B =: C = (c_{ij}), \qquad i = 1, \dots, n, \qquad j = 1, \dots, p$$

$$c_{ij} := \; a_{i1}b_{1j} + \cdots + a_{im}b_{mj} = \sum_{k=1}^{m} a_{ik}b_{kj} \,, \qquad i = 1, \dots, n, \qquad j = 1, \dots, p$$

1. Restriction to square matrices: $n = m$
2. Implement the matrix multiplication for the CPU (loop order should match row-major format)
3. Implement the same algorithm for the GPU
   - With 2D grid-striding loops (2D grid and 2D threadblocks)
   - Without any Blocking and Shared Memory, just Global Memory and Registers
4. Implement a second algorithm for GPU
   - With 2D grid-striding loops and Blocking using square tiles in Shared Memory
   - n is restricted to be multiples of the tilesize T
5. Measure the runtimes of the different implementations with different matrix sizes and compare the speedups.

Hints (also see next page):

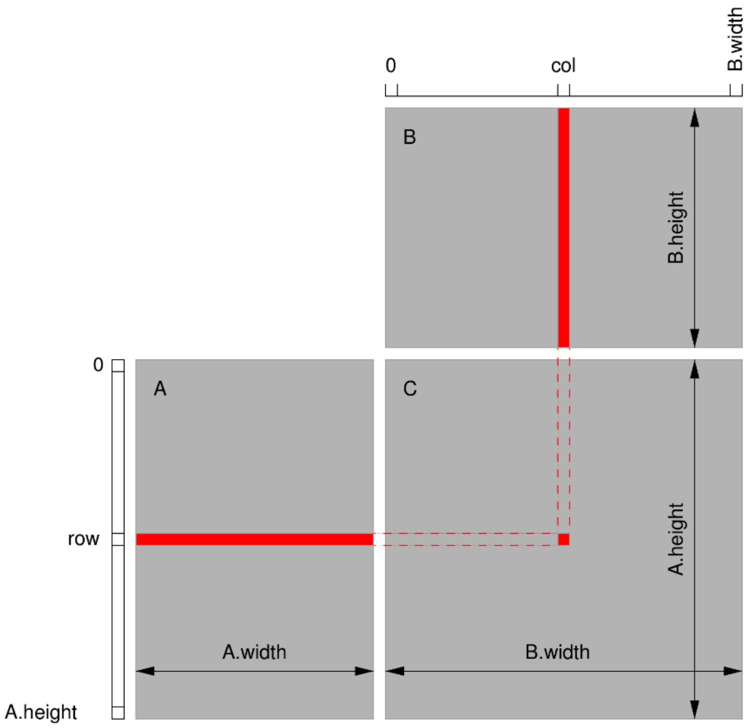
Runtime measurements on GPU can be done with CUDA events:

```
cudaEvent_t start, stop;
float time;
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start, 0);
....
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&time, start, stop);
cudaEventDestroy(start);
cudaEventDestroy(stop);
```

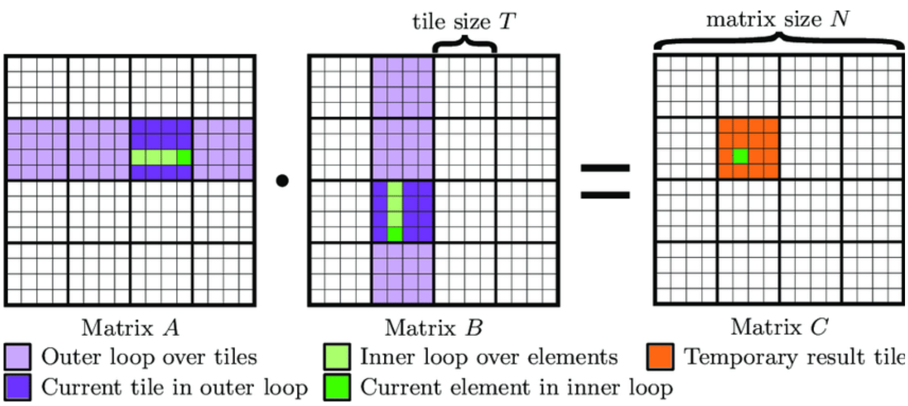
The profiler `nvprof [--print-gpu-trace]` also displays kernel and memcopy runtimes.

"The profilers do change a few behaviors:
- disable some power management when capturing PM (performance metric) counters
- increase the GPU timer frequency from 1 MHz to 31.25 MHz
- measure kernel execution time more precisely than is possible with CUDA events
- increase CPU overhead

- flush work to GPU faster (using a Windows, not Linux difference)" (https://github.com/tdd11235813/cuda-stride-benchmark)
- Matrix multiplication layout



- Matrix multiplication with tiles (blocking) [https://www.researchgate.net/figure/Performance-critical-A-B-part-of-the-GEMM-using-a-tiling-strategy-A-thread-iterates_fig1_320499173]:



- Atomics are not necessary, because the dot product is element-wise
- Do not forget to invoke `__syncthreads()` to synchronize threads of a block (e.g. after shared memory write access)
- Verify the result with the one from the CPU implementation (consumes some amount of time of course)
- Benchmark with higher number of blocks used for grid-striding