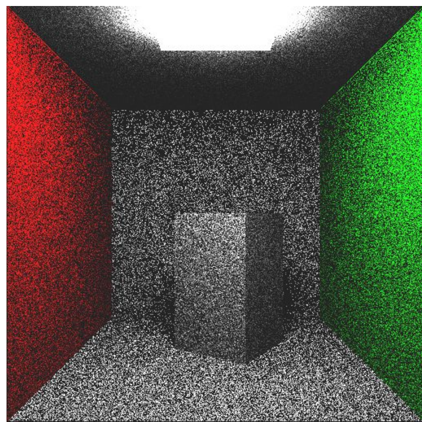
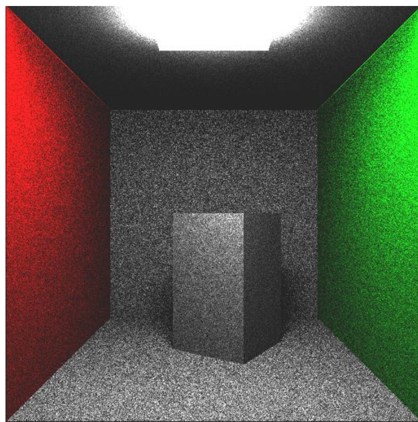
Real time denoising on GPU

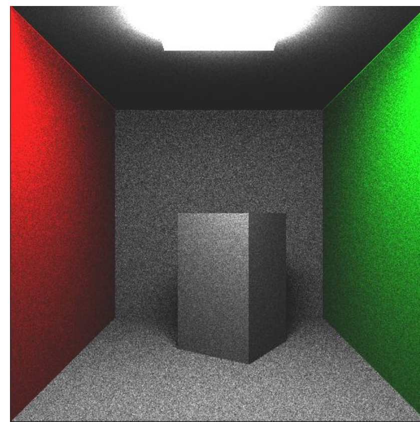
Adrien Vannson. Internship supervised by Johannes Hanika.

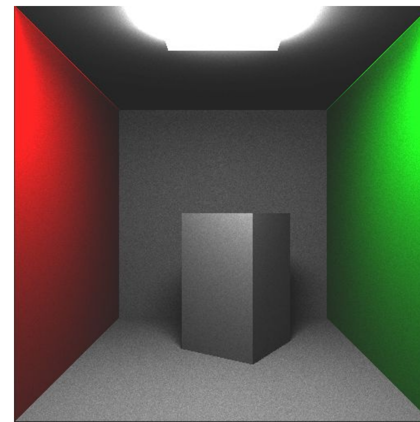
Karlsruher Institut für Technologie

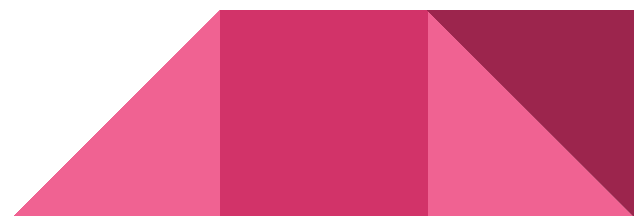
Introduction


100 samples per pixel


300 samples per pixel


900 samples per pixel


10000 samples per pixel

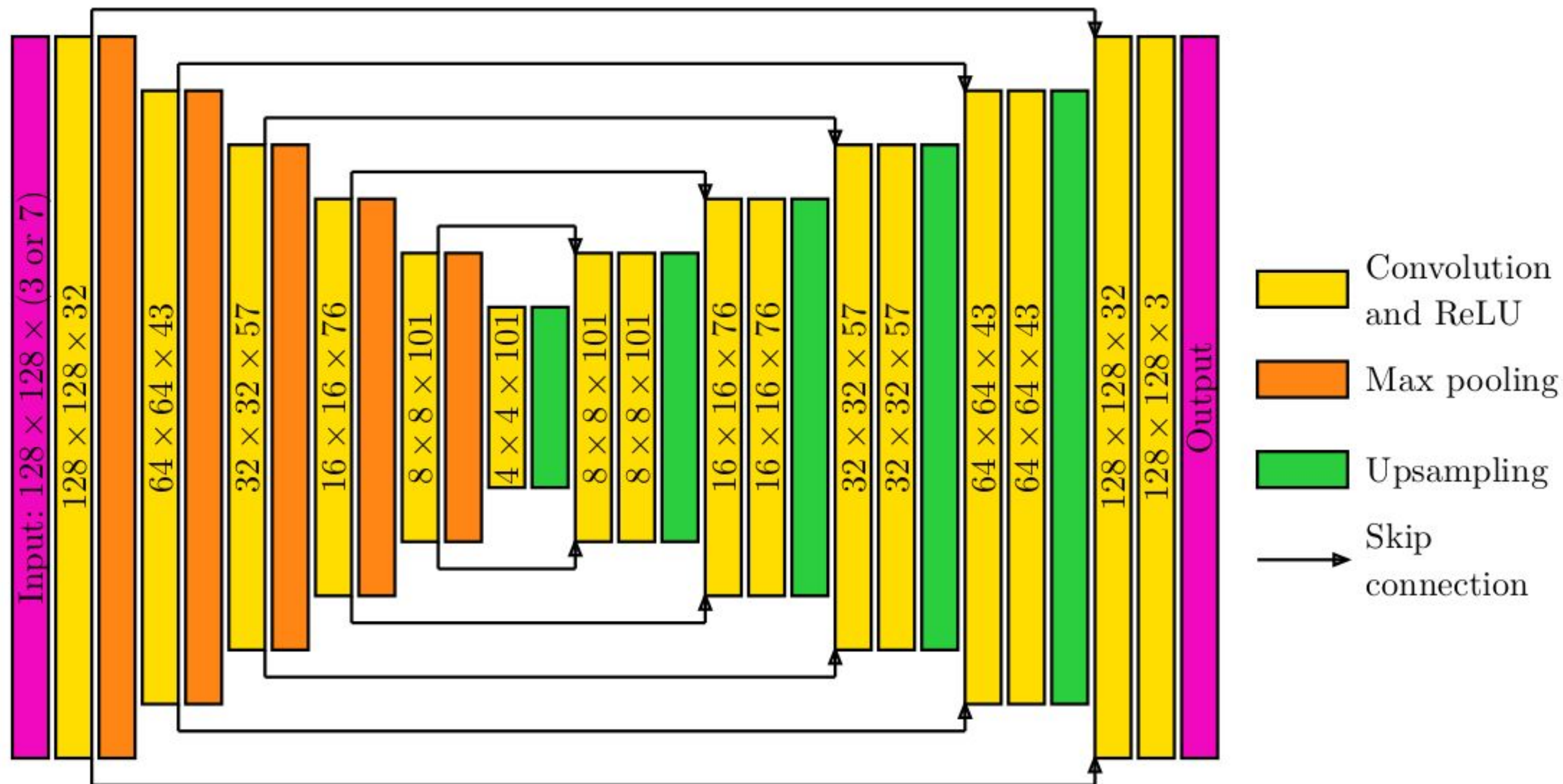


Motivation

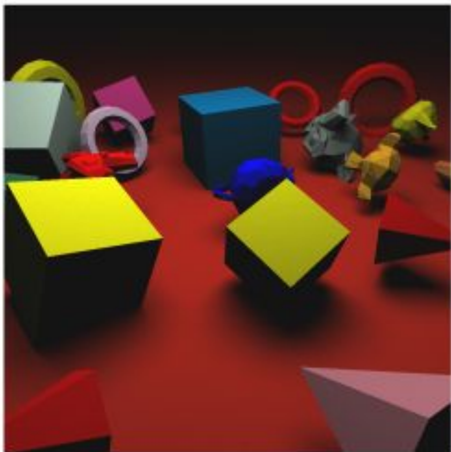
- Path tracing: promising but unusable in real time
- ML denoising: can work but costly to execute
  - ML frameworks are not designed to work at real-time rates
  - Libraries offering efficient matrix multiplication algorithms often need CUDA
  - Difficult to use the Tensor Cores with Vulkan
- NVIDIA released the Cooperative Matrices extension to allow using Tensor Cores
- Does it allow us to reach real-time rates for ML denoising?

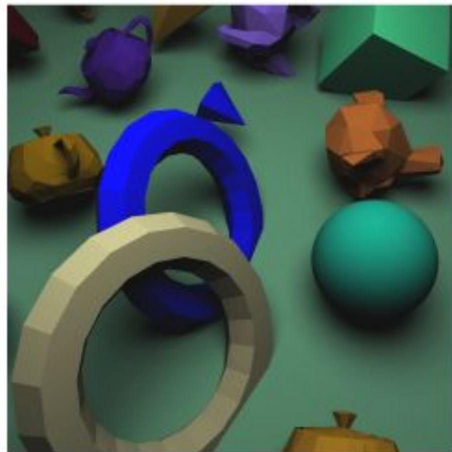
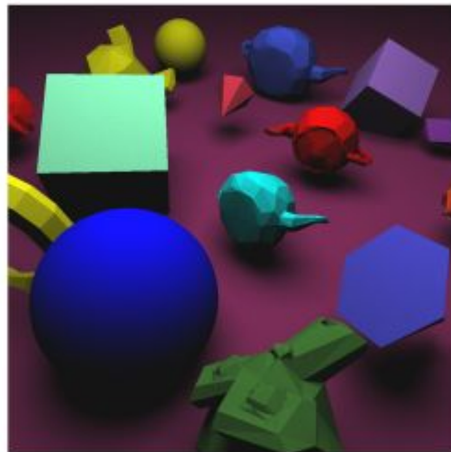
Denoising images

Architecture of the U-Net network

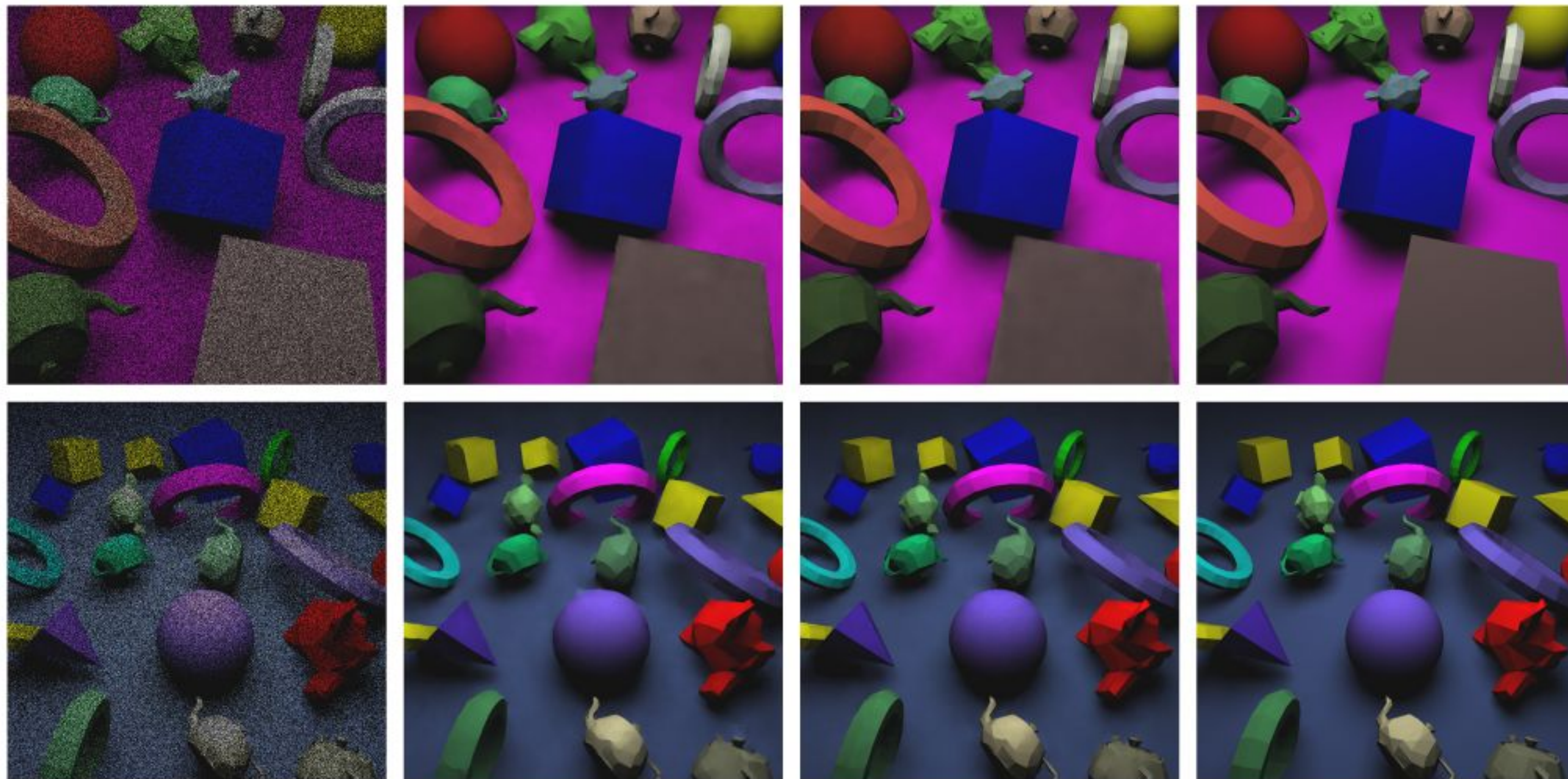


Training

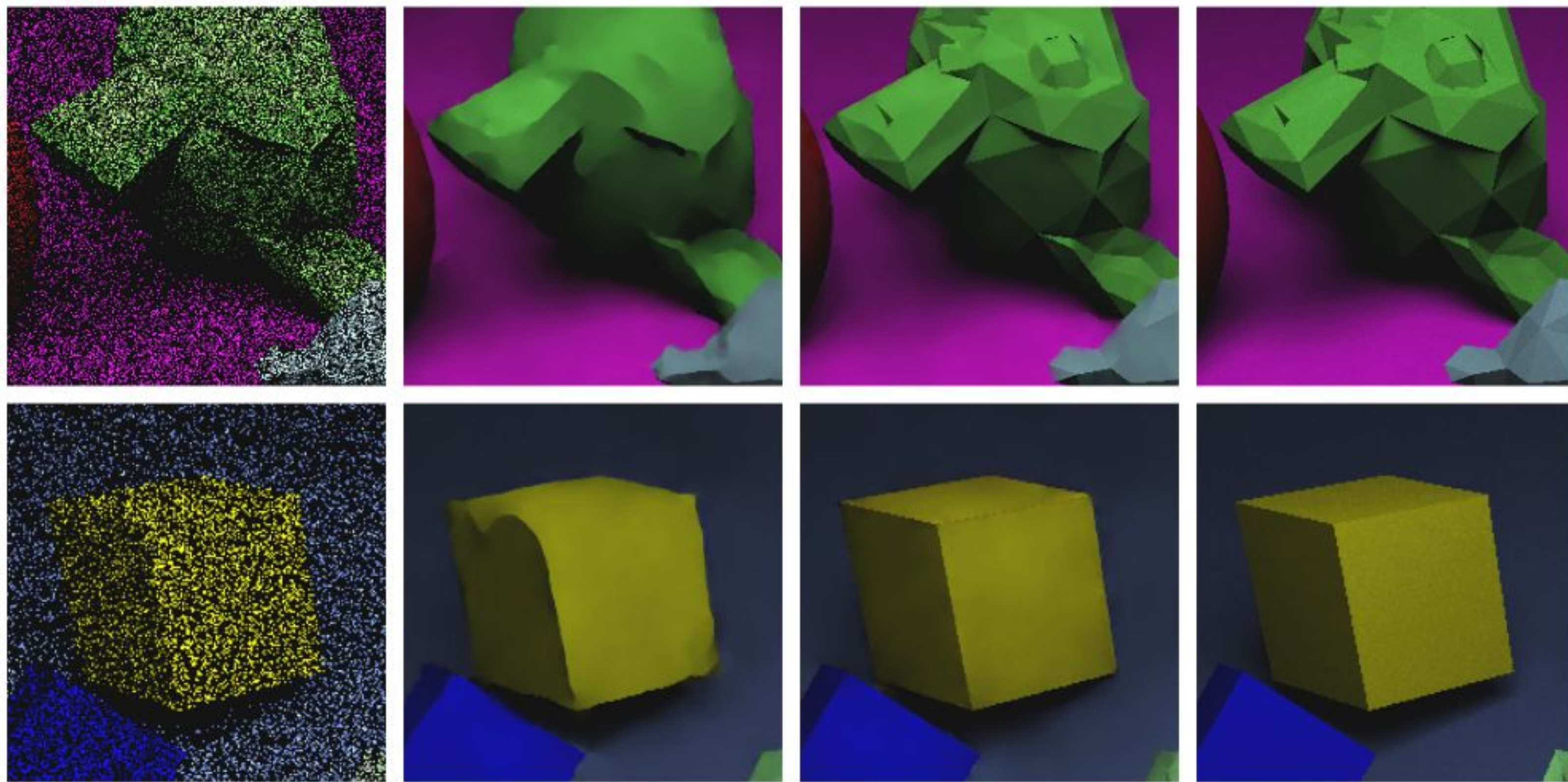


Evaluation



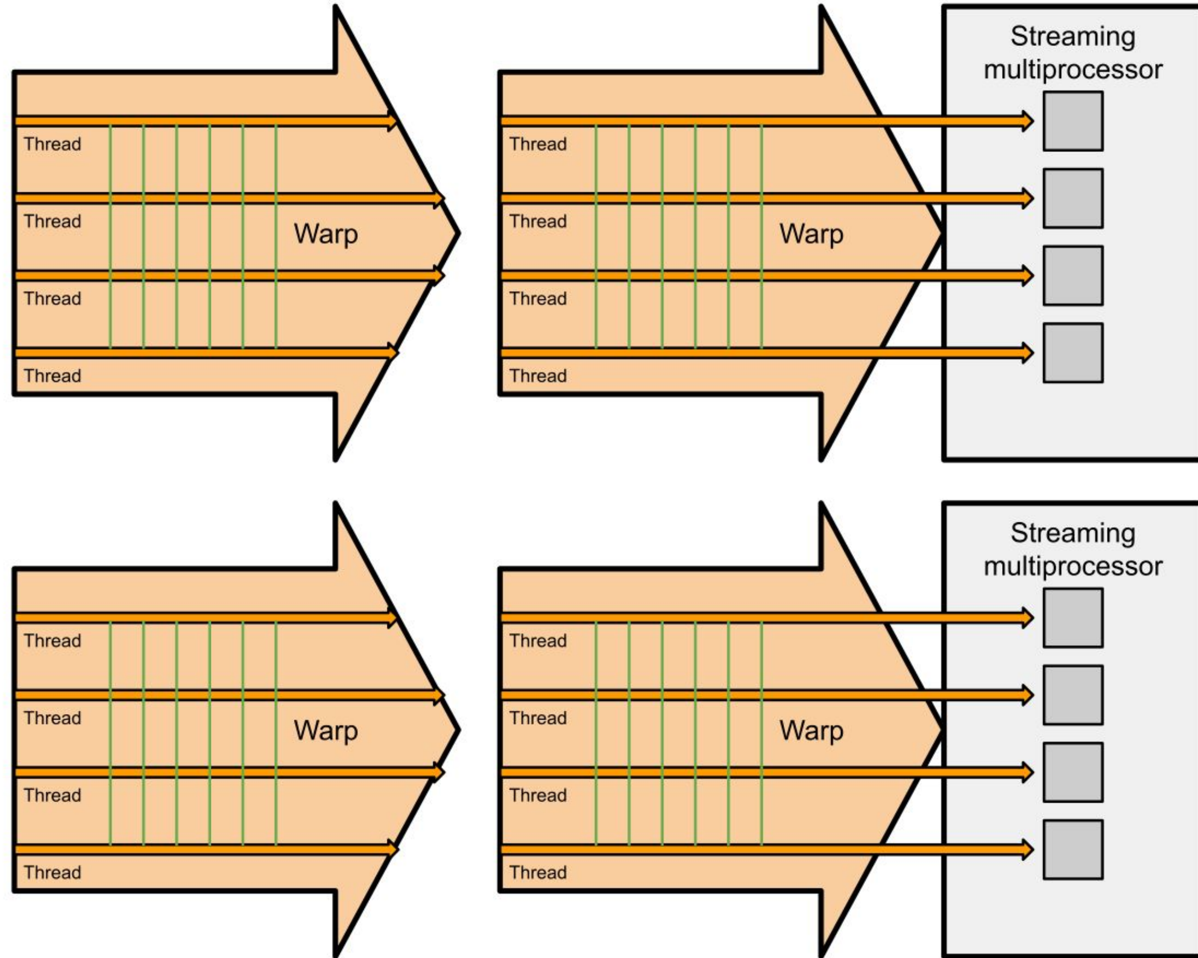
Evaluation



Introduction to GPU computing

Thread hierarchy

- Programs: composed of *shaders*, ie files defining a function called the *kernel*.
- Kernel: executed several times in parallel, each instance is a *thread*.
- The threads are grouped in *thread blocks* (programming abstraction) and *warps* (low-level).
- A warps contains 32 threads executed together (SIMT)

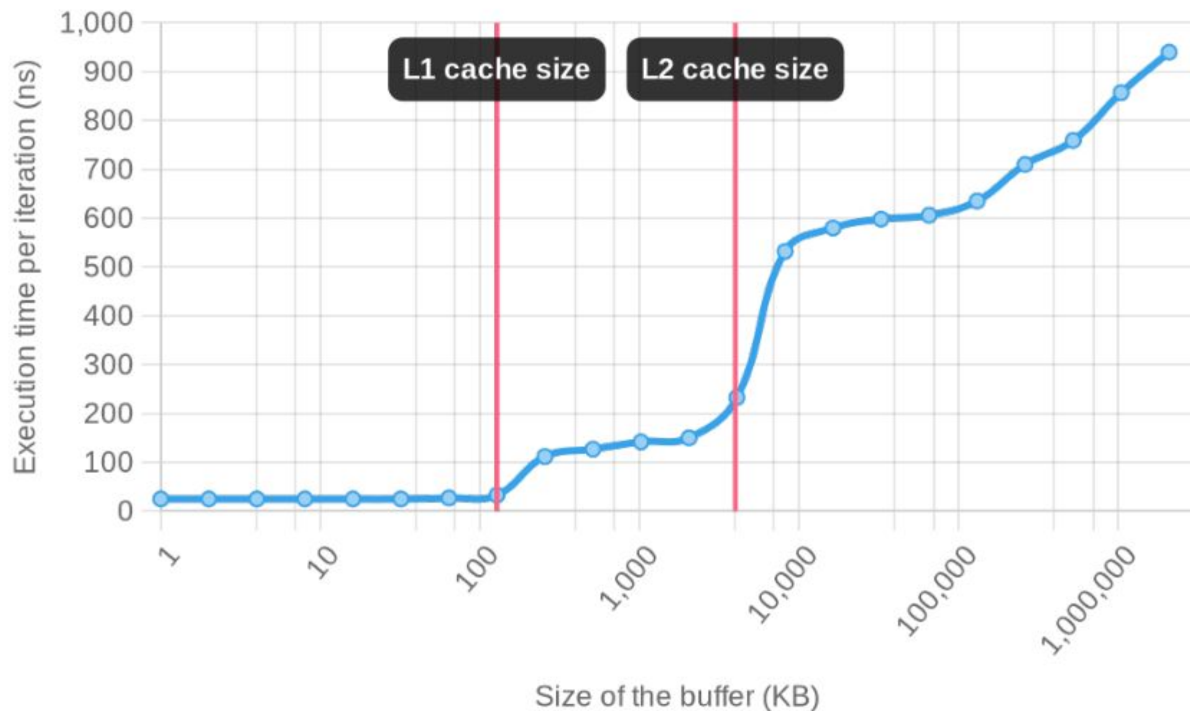


Memory hierarchy

- Global memory
  - Compares to the RAM for CPU
  - Accessible by all the threads
  - ≈ 10 GB
  - Very slow (≈500 ns latency)
- Shared memory
  - Located on the SM → accessible by the threads of the same thread block
  - ≈ 100 KB
  - Fast (≈1 ns latency)
- Registers
  - Located on each processor, can not be shared among threads
  - Fastest memory available

Hardware effects: global memory

```
uint pos = 0;
for (int i = 0; i < 1000000; i++) {
    pos = buffer[pos];
}
```

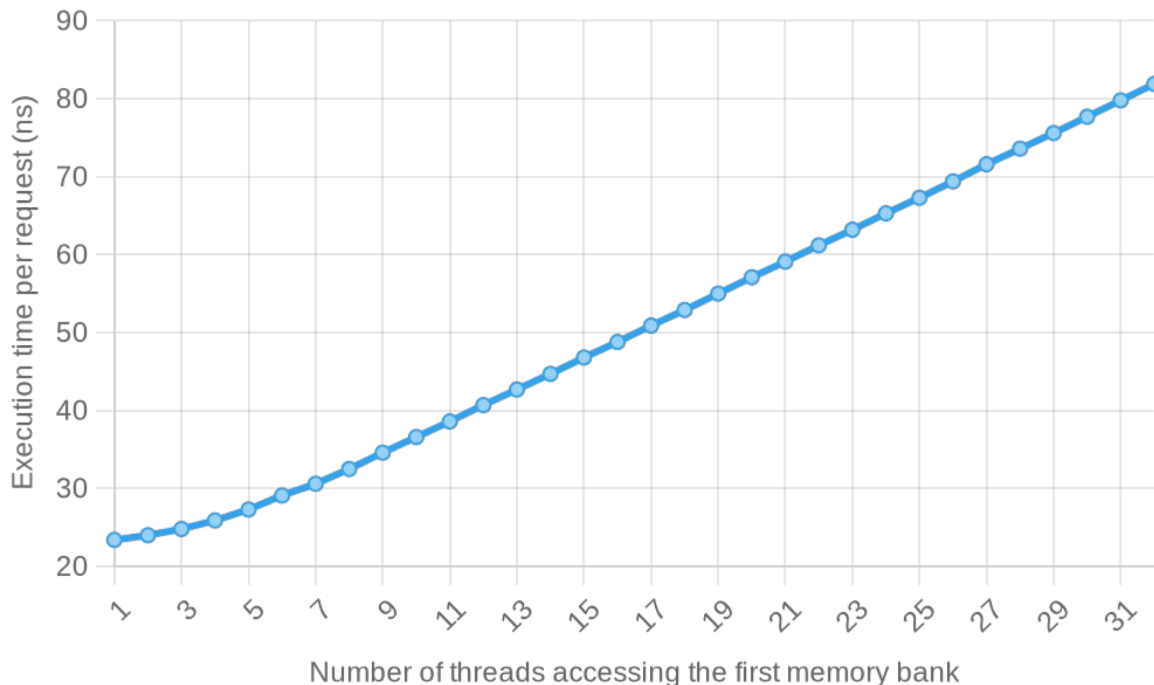


Hardware effects: memory coalescing

- When the threads of a warp need global memory, their requests are treated together.
- If possible (ie contiguous memory requested), only one transaction is performed.
- Otherwise, multiple transactions are realized → slower

“The concurrent accesses of the threads of a warp will coalesce into a number of transactions equal to the number of 32-byte transactions necessary to service all of the threads of the warp.”

Hardware effects: bank conflicts



- Shared memory: stored on 32 components per SM
- If all the threads need memory on the same component: bank conflict → slow request

Hardware effects: thread divergence

- SIMT (Single Instruction, Multiple Threads) for threads of the same warp

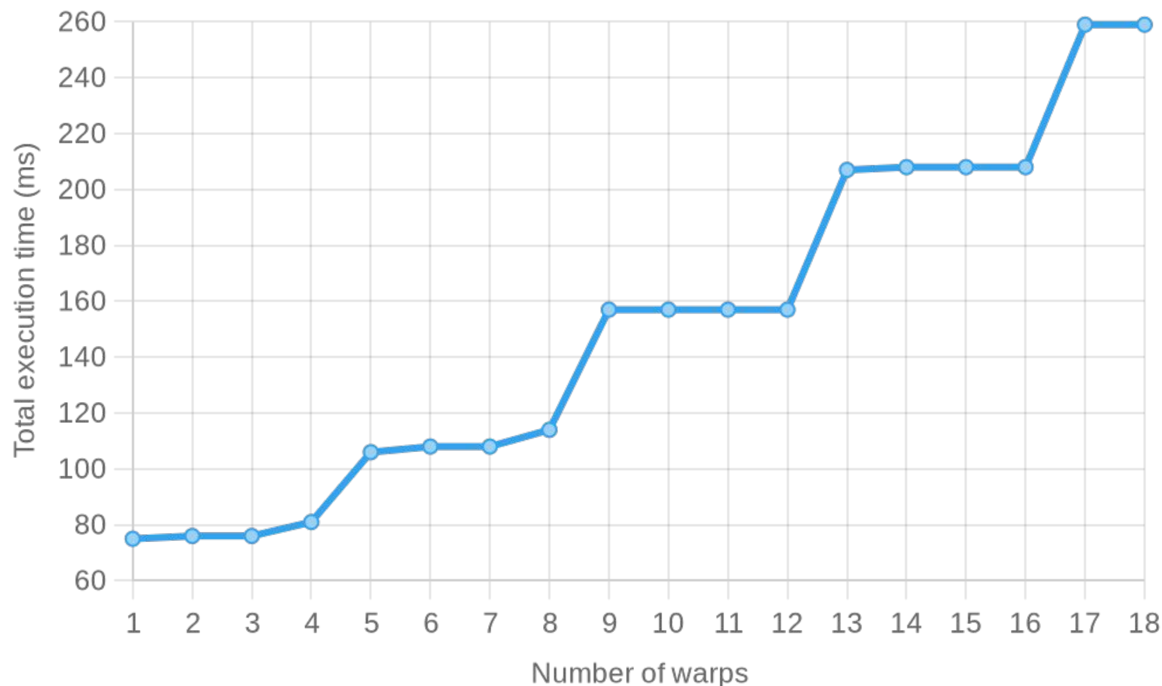
➢ The same instruction is performed by all the threads

- What about `if` statements?
- Some threads are disabled
- In a `if` / `else` statement: both blocks are executed serially

```
if C:
    f()
else:
    g()
```

Matrix multiplication

Tensor cores



- Used to multiply 16 x 16 matrices
- 4 tensor cores by SM

➢ Schedule at least 4 x 32 threads per SM

Maximal throughput:

➢ **47 TFlops**

Naive algorithm

- $A \in \mathbb{R}^{n \times p}$, $B \in \mathbb{R}^{p \times m}$
- Naive $O(n \times p \times m)$ algorithm
- One thread for each output coefficient
- Read everything from global memory
- 0.17 TFlops

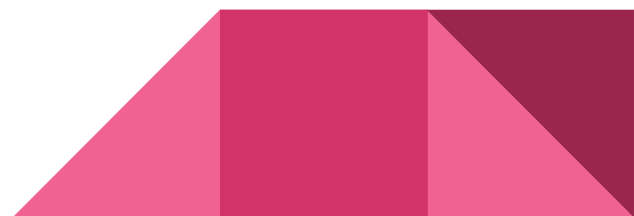
$$\sum_{k=1}^{p} A_{i,k} B_{k,j}$$

Shared memory

- One thread for each output coefficient, 16 × 16 thread blocks
- Load 16 × 16 tiles to shared memory

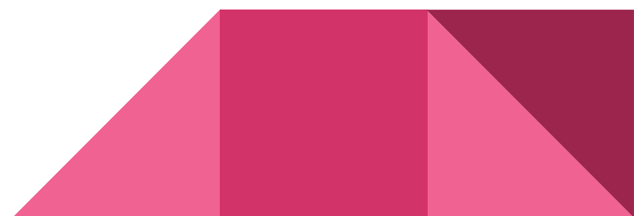
➢ Only p / 16 reads per coefficient

- 0.81 TFlops



Cooperative matrices

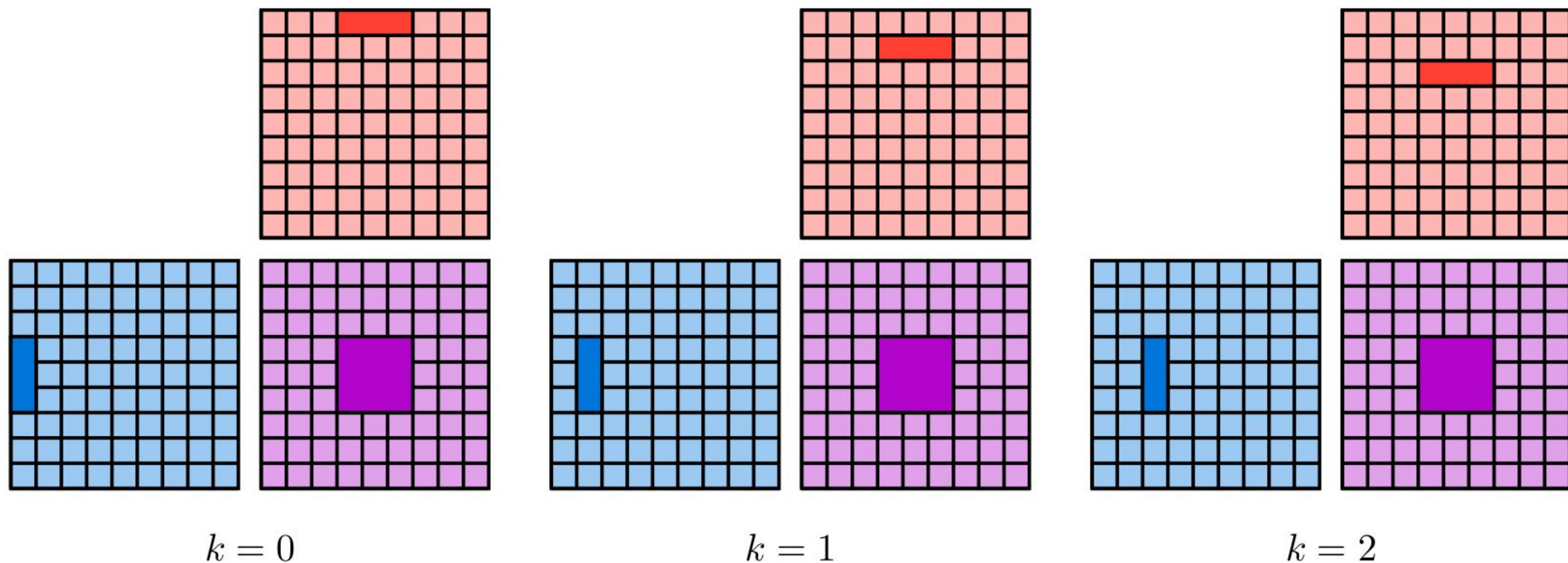
- Same algorithm but using tensor cores to perform the product
- 8.6 TFlops



Efficient use of cooperative matrices

- Data is still loaded too often from global memory
- ➢ Use bigger thread blocks (256 x 256)
- ➢ Use rows / columns instead of tiles

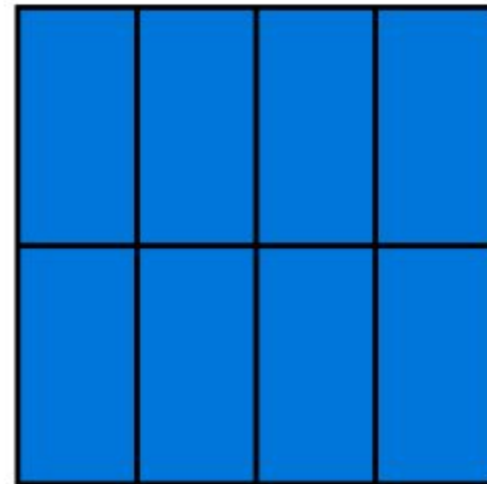
Efficient use of cooperative matrices



Efficient use of cooperative matrices

- Lots of memory needed → only one thread block per SM
- To use all the four tensor cores, we need more than one warp
- 8 warps (ie 8 x 32 = 256 threads) per block

✓ 34 TFlops!



Performance evaluation

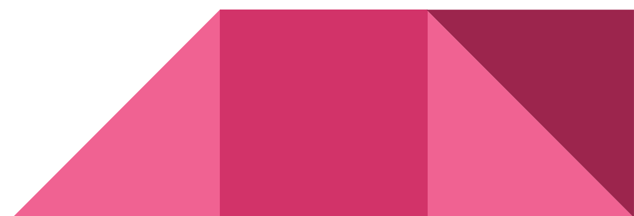
Theoretical upper bound

Throughput (TFlops)

50
45
40
35
30
25
20
15
10
5
0

Naive
Shared memory
Cooperative matrices
Tiled

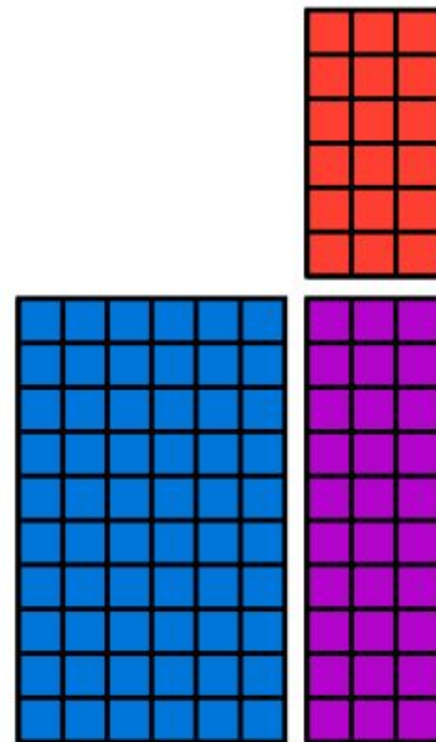
➢ We can use the tensor cores very efficiently!



Fast network inferencing

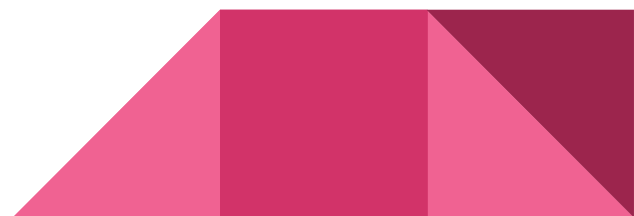
Convolution as matrix product

- Matrix A, in blue: input data
- Matrix B, in red: weights
- Matrix C, in purple: output data



Inferencing the network

- One shader per convolution: max pooling, upsampling and skip connections do not have their own shader
- Shaders generation at compile time → more compiler optimizations



Performance evaluation

Convolution	0	1	2	3	4	5
Time (ms)	0.93	2.31	0.99	0.83	0.99	0.51
TFlops	2.27	1.41	1.46	0.77	0.28	0.18

Table 1: Performance of the encoder

Convolution	6	7	8	9	10	11	12	13	14	15
Time (ms)	0.84	0.35	1.17	0.82	2.82	1.2	3.87	1.68	8.16	3.32
TFlops	0.9	2.17	1.69	2.08	1.59	3.18	2.62	5.2	2.78	0.55

Table 2: Performance of the decoder

Conclusion