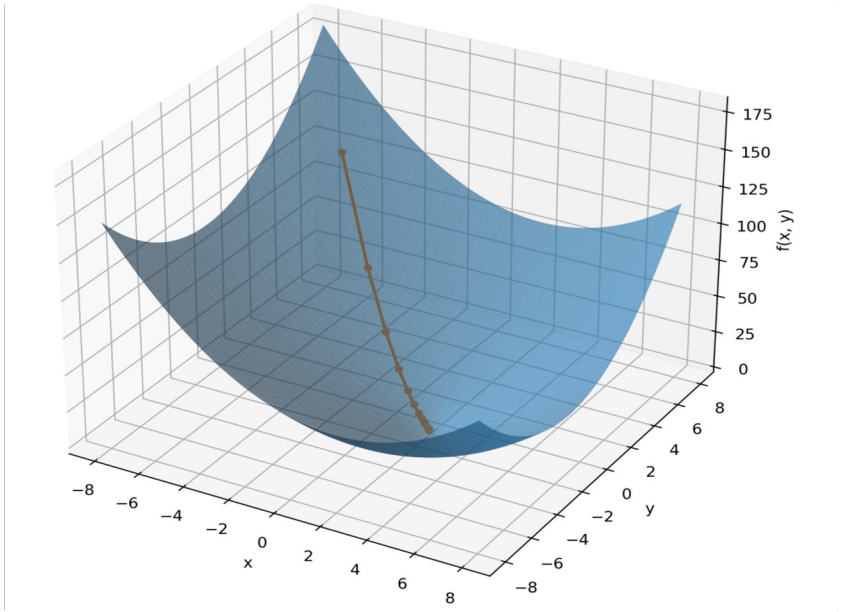
Performance improvements of commonly used optimizers in machine learning using **CUDA** and **OpenCL**

Lennart Hahner

Motivation

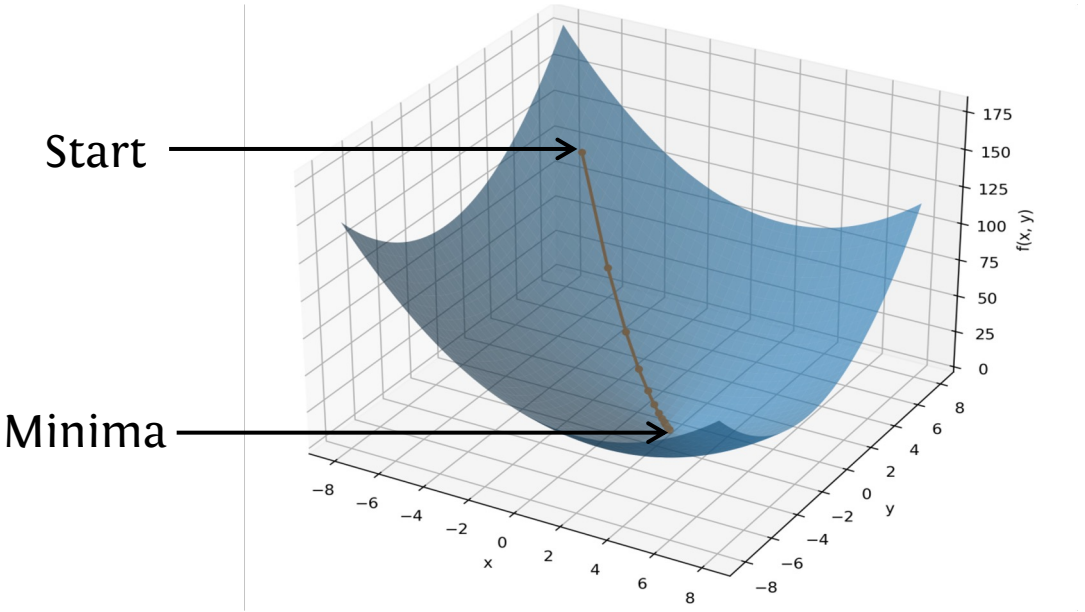
The role of optimization in machine learning

Optimizers are the engines of machine learning models.



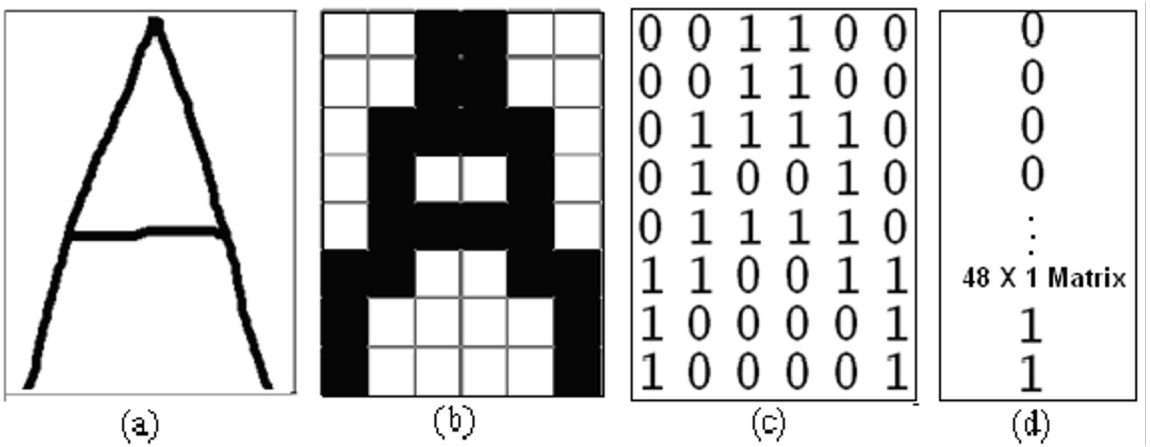
The role of optimization in machine learning

Train learnable weights towards a minimum of a loss function.



The role of optimization in machine learning

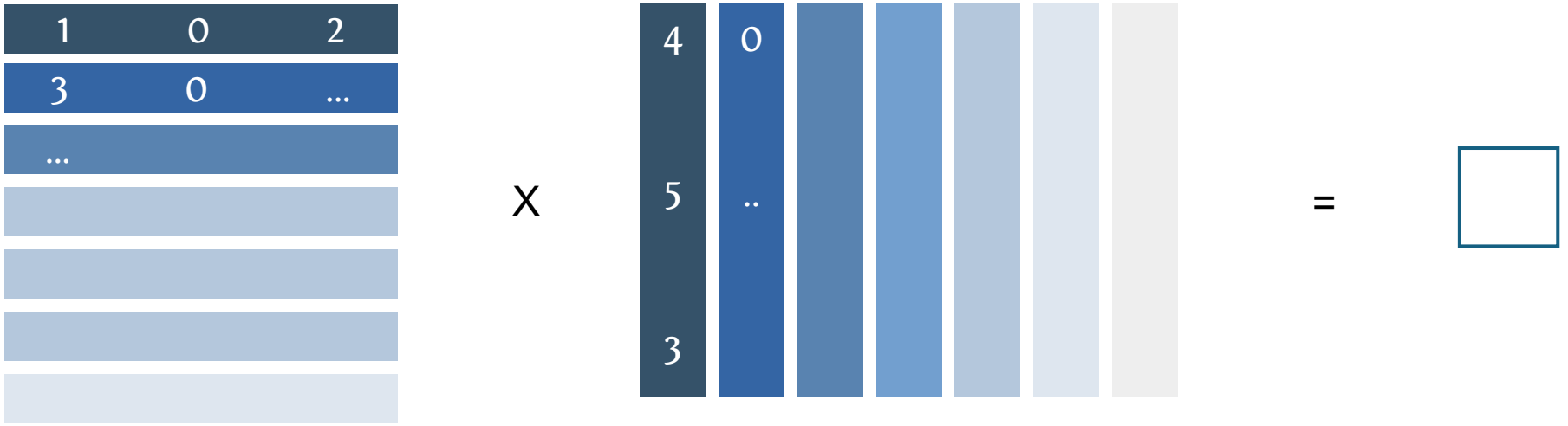
Most inputs e.g. images or text are represented as matrices...



[1] Figure 2 from Improving the Character Recognition Efficiency of Feed Forward BP Neural Network

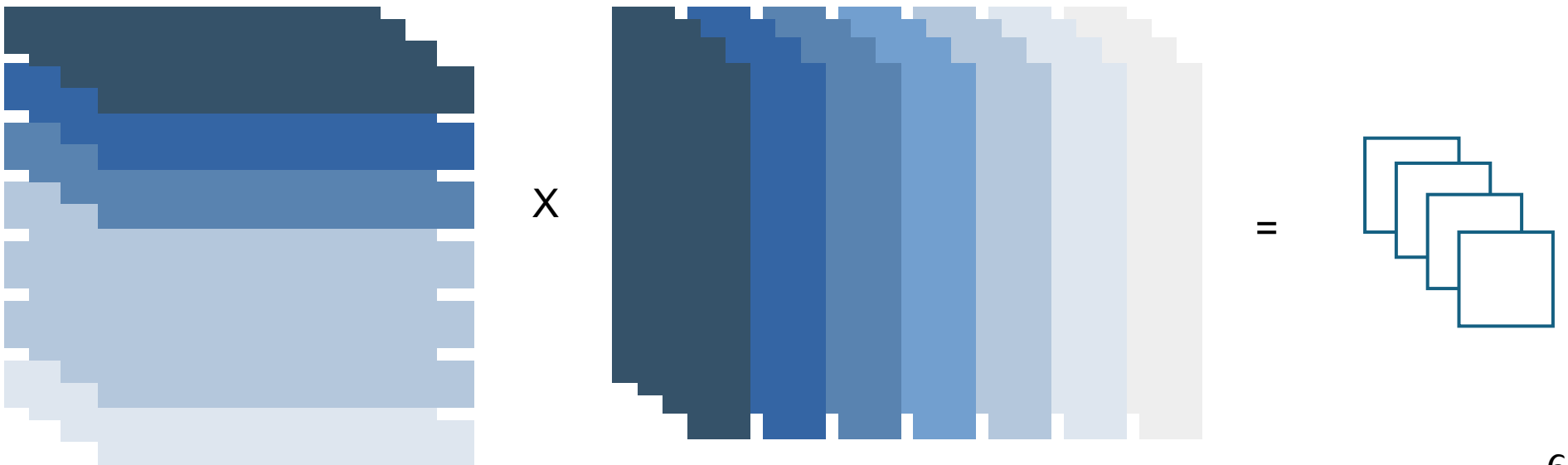
The role of hardware in machine learning

Thus, we often deal with **matrix** or tensor operations during optimizations...



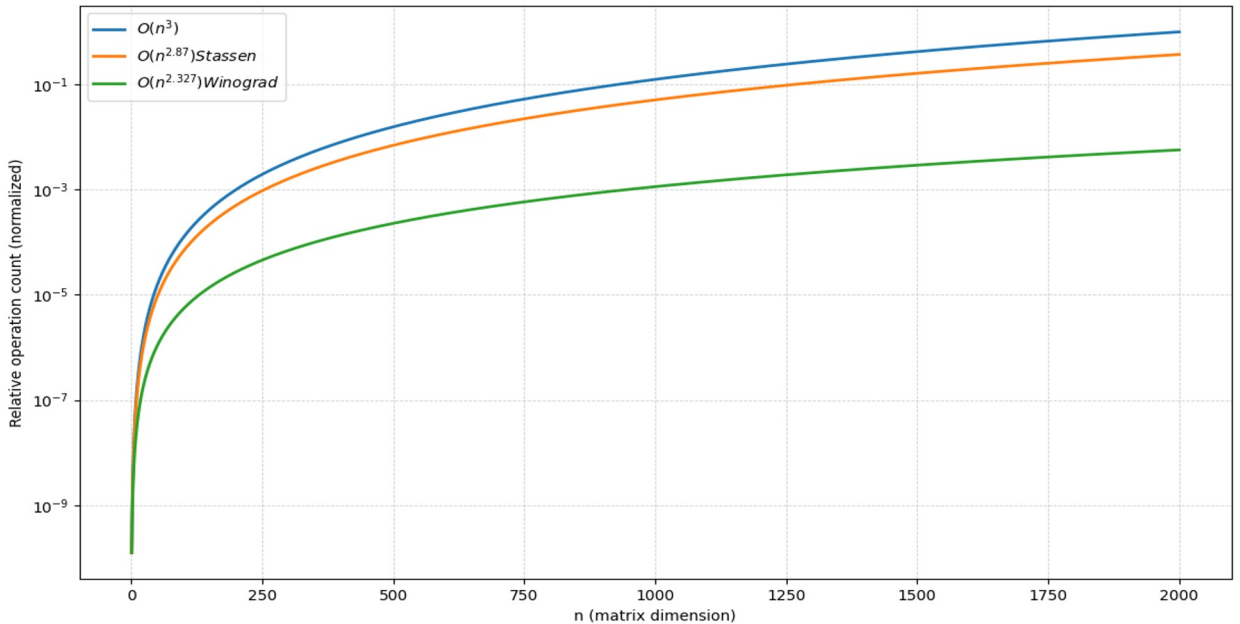
The role of hardware in machine learning

Thus, we often deal with matrix or **tensor** operations during optimizations...



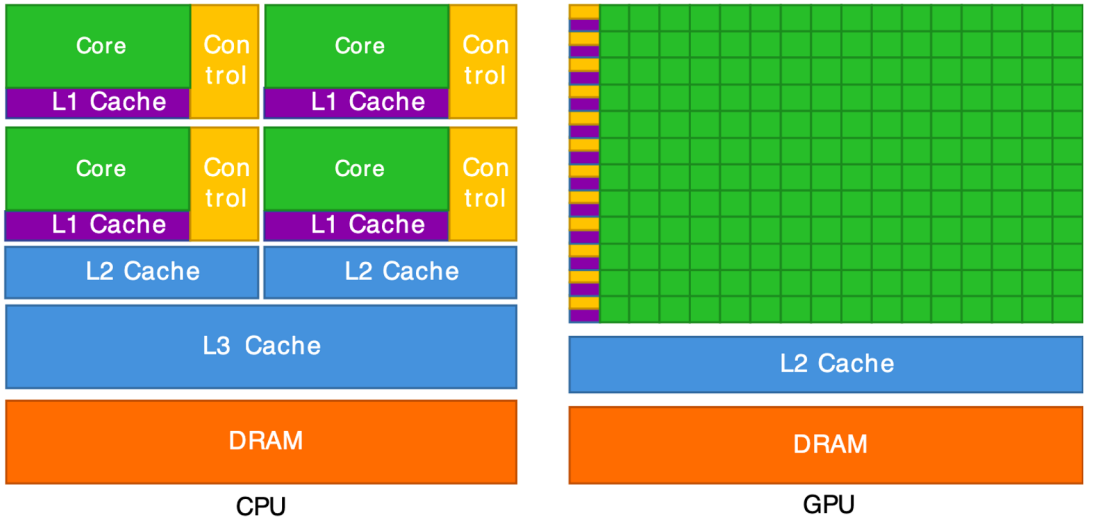
The role of hardware in machine learning

Matrix and Tensor operations are computationally expensive...



The role of hardware in machine learning

...exploiting hardware for parallelization



[2] Figure 2 - Review of GPU-based Monte Carlo simulation platforms for transmission and emission tomography in medicine.

The role of hardware in machine learning

...exploiting hardware for parallelization

Specification	**Intel Core i9 14900KF** [3]	**Nvidia GeForce RTX 5080** [4]
Cores	24	16384+
Max. Boost Clock (Ghz)	6	2.62
Architecture	x86	Blackwell
Core Paradigma	MIMD	SPMD

[3] Intel
[4] NVIDIA

The role of hardware in machine learning

...how to exploit GPUs?

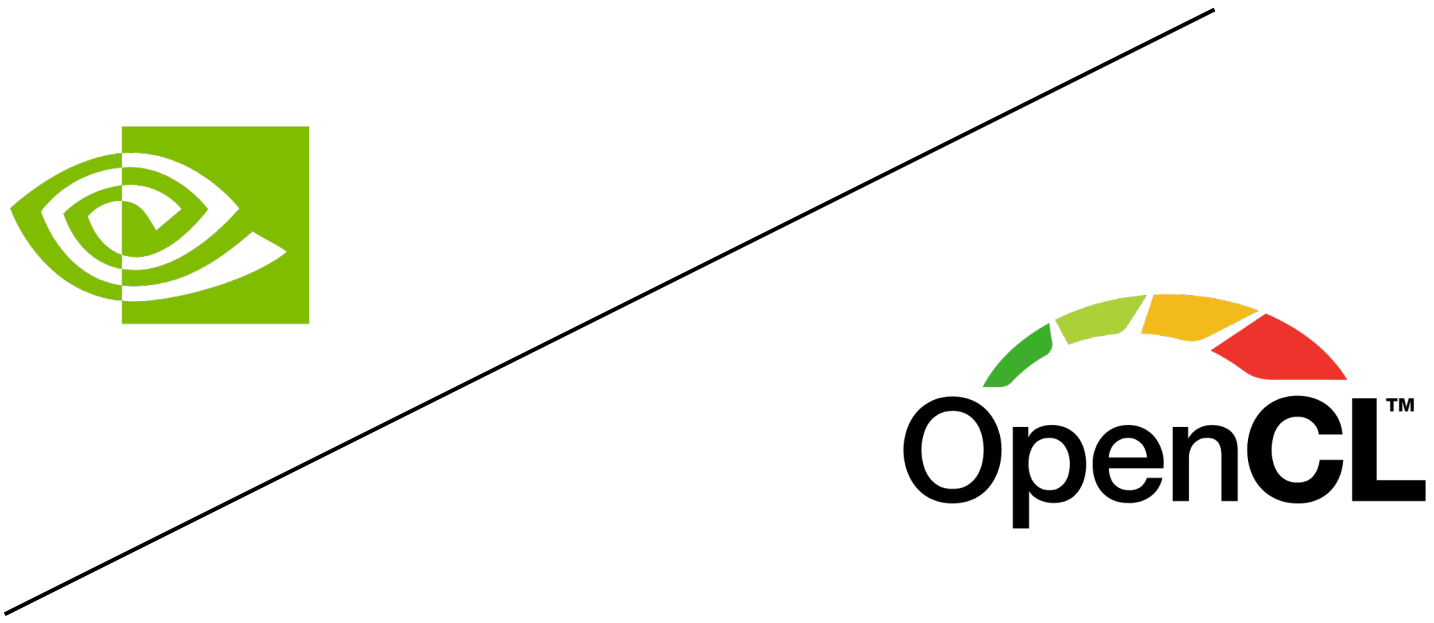


Table of Contents

Expectations

Results of „From CUDA to OpenCL“ [5]

OpenCL enables code portability, but not performance

OpenCL GEMM program

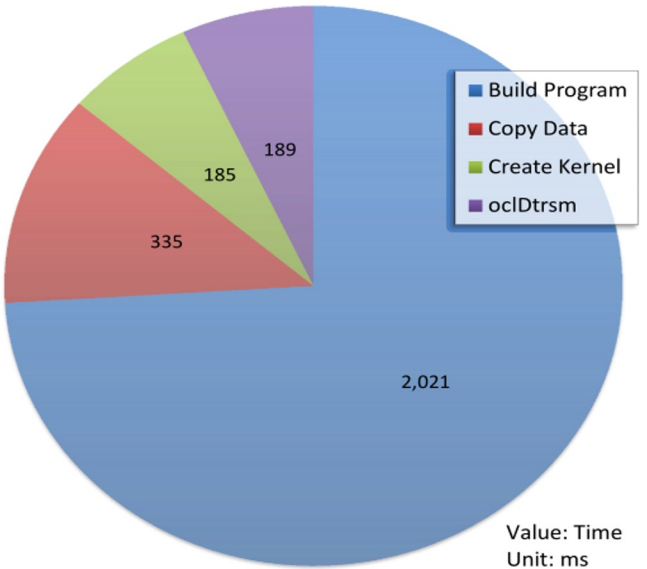


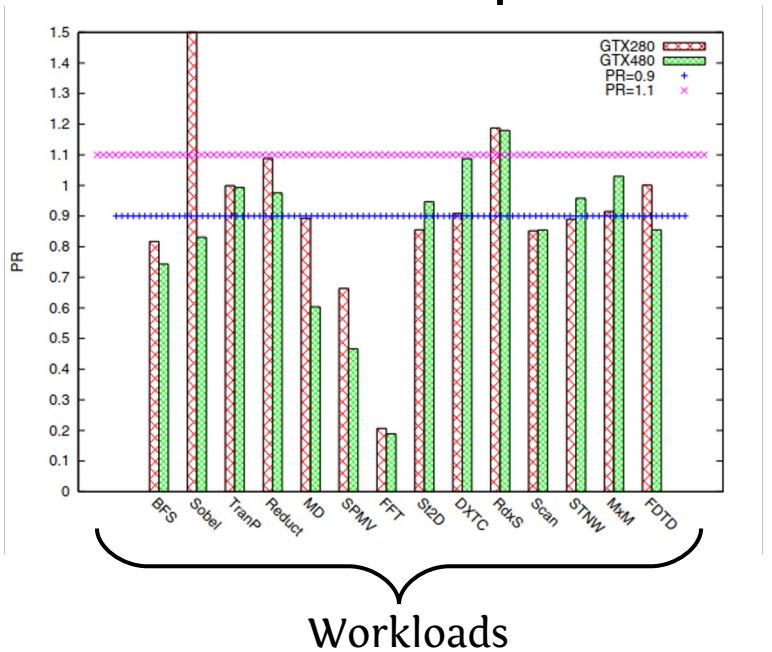
Figure 3: Runtime break down

[5] „From CUDA to OpenCL: Towards a Performance-portable Solution for Multi-platform GPU Programming“ Peng D.

Results of „A comprehensive Performance comparison of CUDA and OpenCL“ [6]

CUDA is at most ~30% faster than OpenCL for most applications

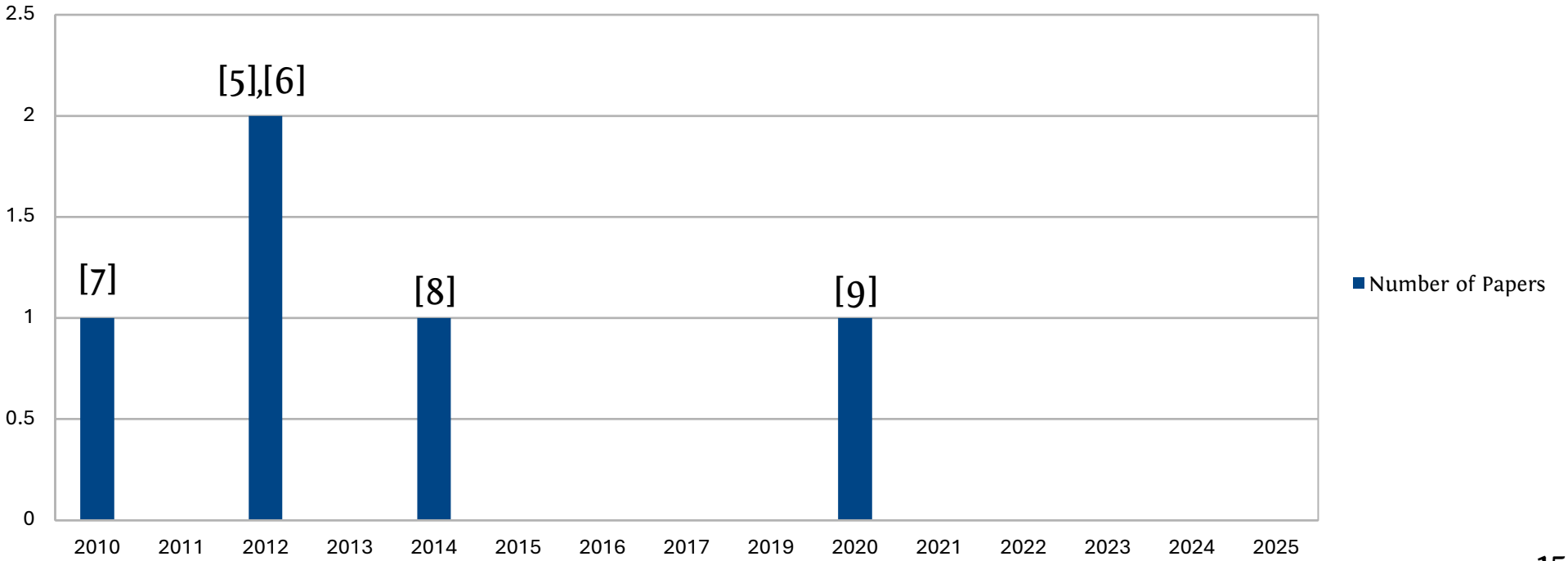
$$PR = \frac{Perf(OpenCL)}{Perf(CUDA)}$$



[6] J. Fang, A. L. Varbanescu and H. Sips, "A Comprehensive Performance Comparison of CUDA and OpenCL,"

Literature review

Most of the comparison results are more than 10 years old...

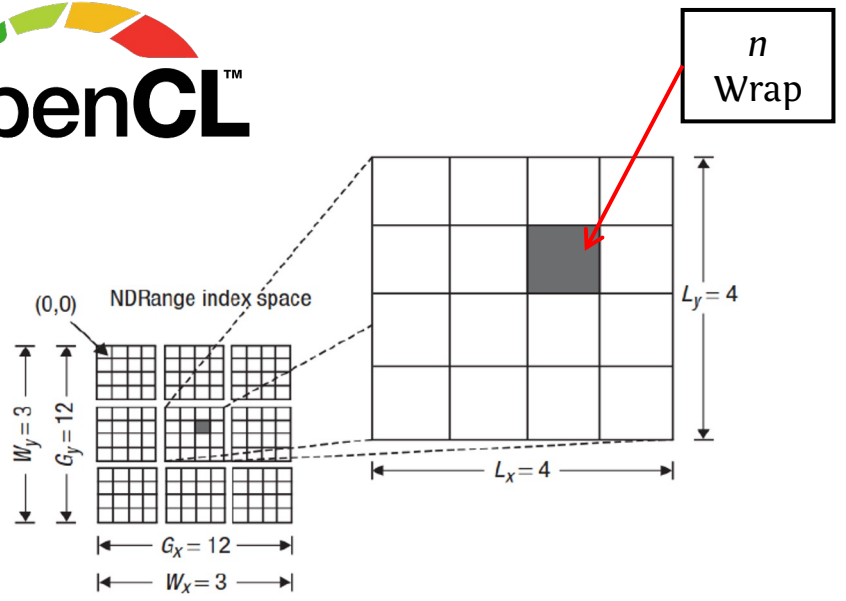


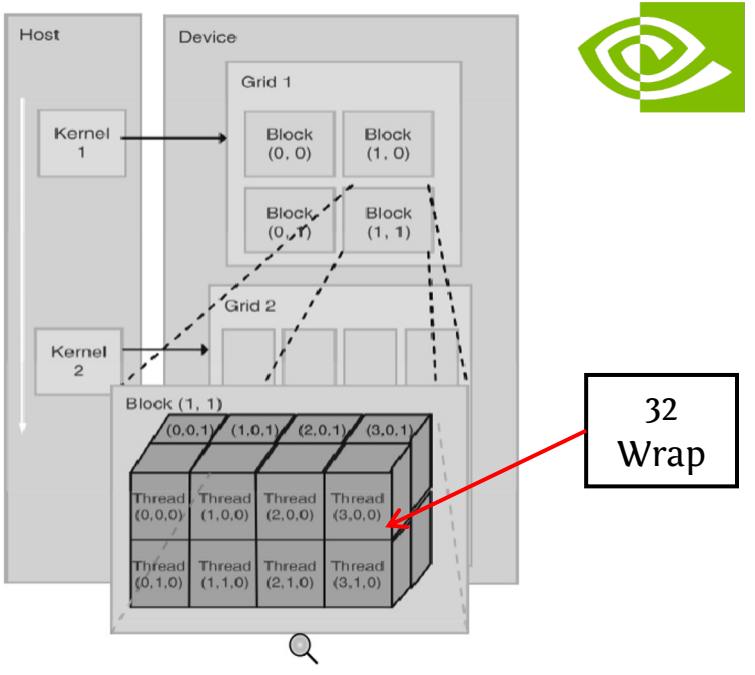
So, what to expect from the comparison?

...CUDA might perform better, due to the longer compile/build time of OpenCL, but it's worth revisiting...

Conceptual comparison of CUDA and OpenCL

Abstraction differences

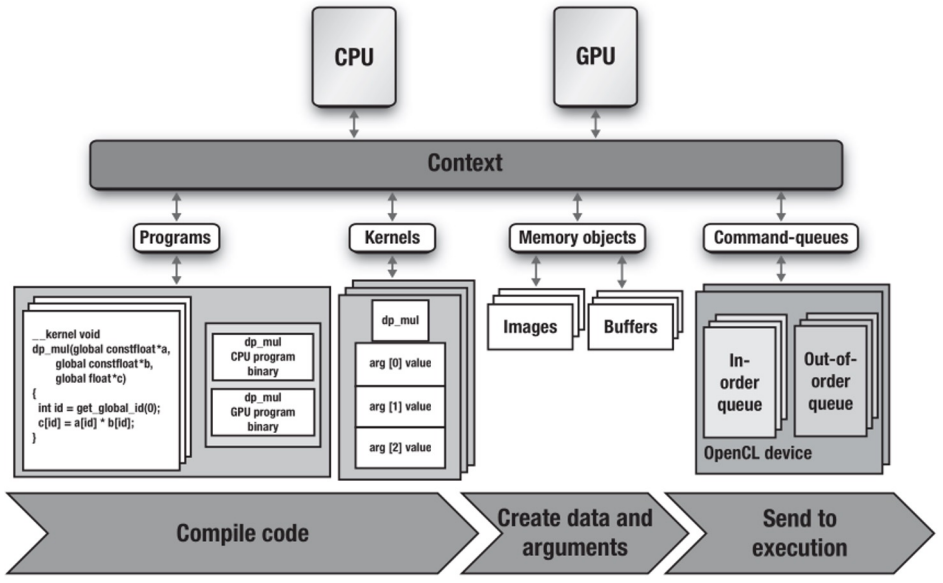


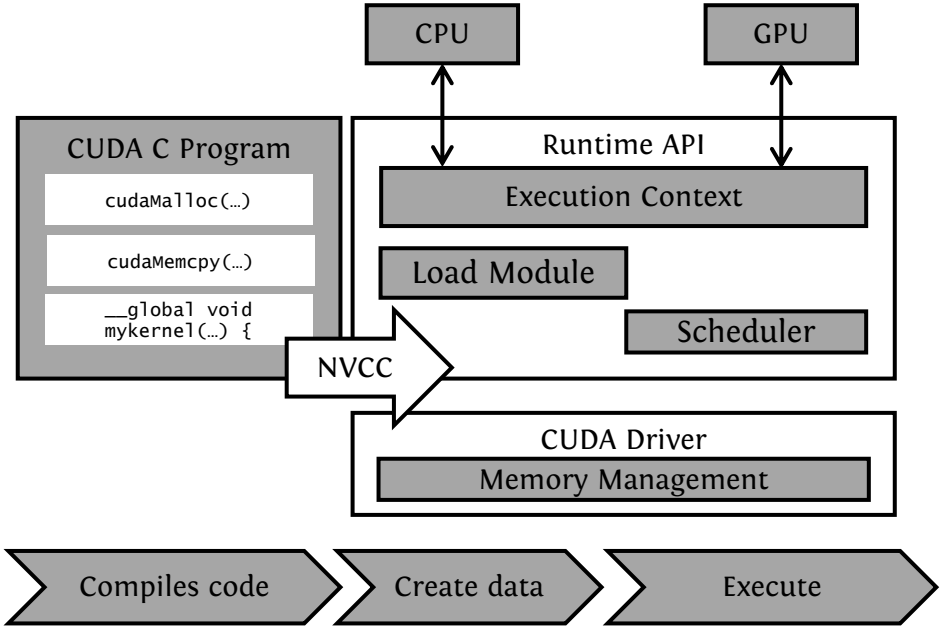


[10] Figure 1.7 in Aftab M. et al. "OpenCL Programming Guide"

[11] Figure 3.13 in Kirk D. et al. "Programming Massively Parallel Processors""

Architecture differences





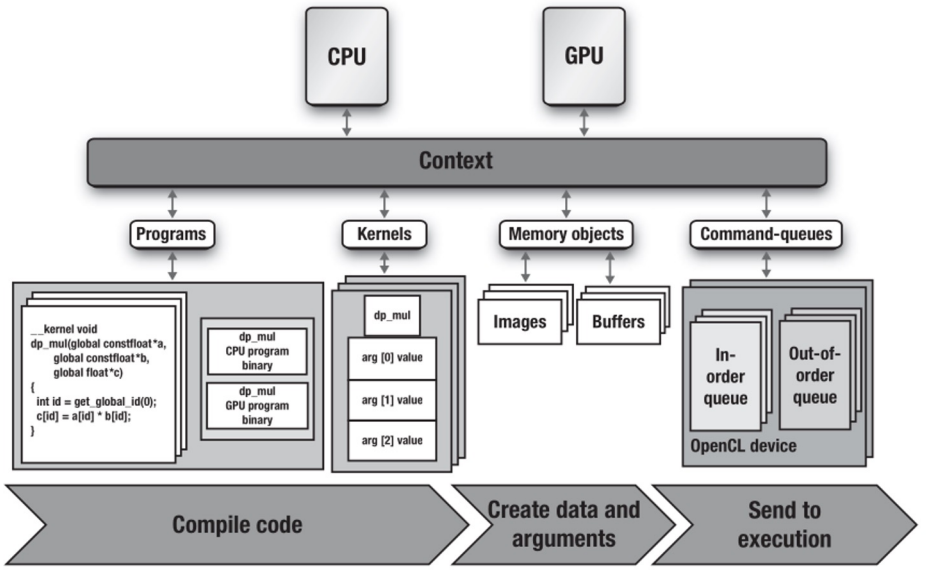
[10] Figure 1.7 in Aftab M. et al. "OpenCL Programming Guide"

[12] Based on CUDA Programming Guide

Architecture differences

Unlike CUDA, OpenCL requires an environmental setup...

1. Get `cl_device_id`
2. Create `cl_context`
3. Create `cl_command_queue`
4. Create `cl_program`
5. Define `cl_kernel`
6. Allocate `cl_mem`



[11] Figure 1.9 in Aftab M. et al. "OpenCL Programming Guide"

Architecture differences

NVCC enables easy integration of kernel functions...

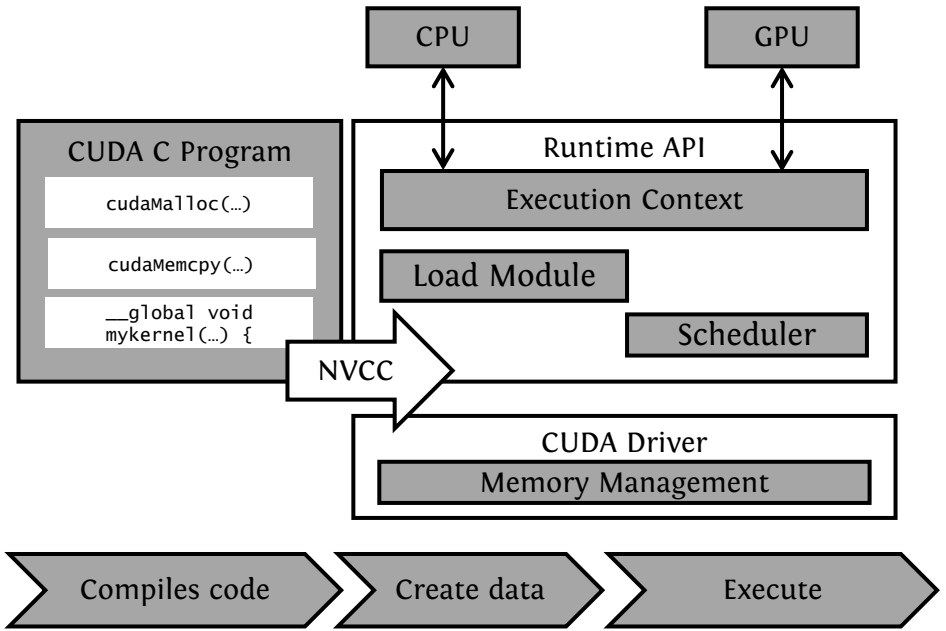
1. Allocate device memory
`cudaMalloc(...)`

2. Copy to device `cudaMemcpy(...)`

3. Launch kernel
`kernel«blocks, threads»(...)`

4. Copy from device `cudaMemcpy(...)`

5. Free `cudaFree(...)`



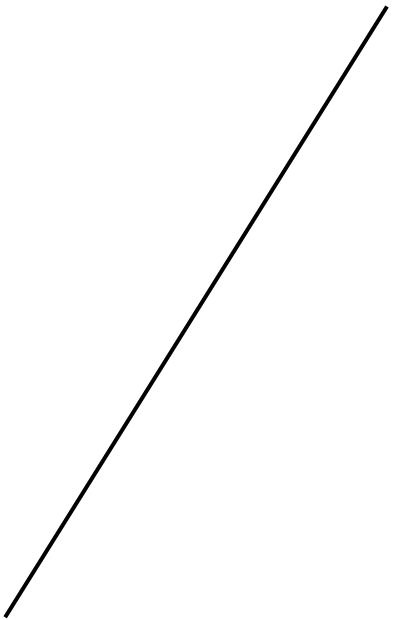
[12] Based on CUDA Programming guide

Kernels in different languages

E.g.: the update step of the Adam optimizer

OpenCL™

```
__kernel void adam_update(
    __global float*       param,
    __global const float* grad,
    __global float*       m,
    __global float*       v,
    const float lr,
    const float beta1,
    const float beta2,
    const float eps,
    const float bc1,
    const float bc2,
    const int   n
){
    ...
}
```



```
__global__ void adam_update_kernel_impl(
        float *__restrict__ param,
        const float *__restrict__ grad,
        float *__restrict__ m,
        float *__restrict__ v,
        size_t n,
        float lr, float beta1,
        float beta2,
        float bc1, float bc2, float eps)
{
    ...
}
```

Summary

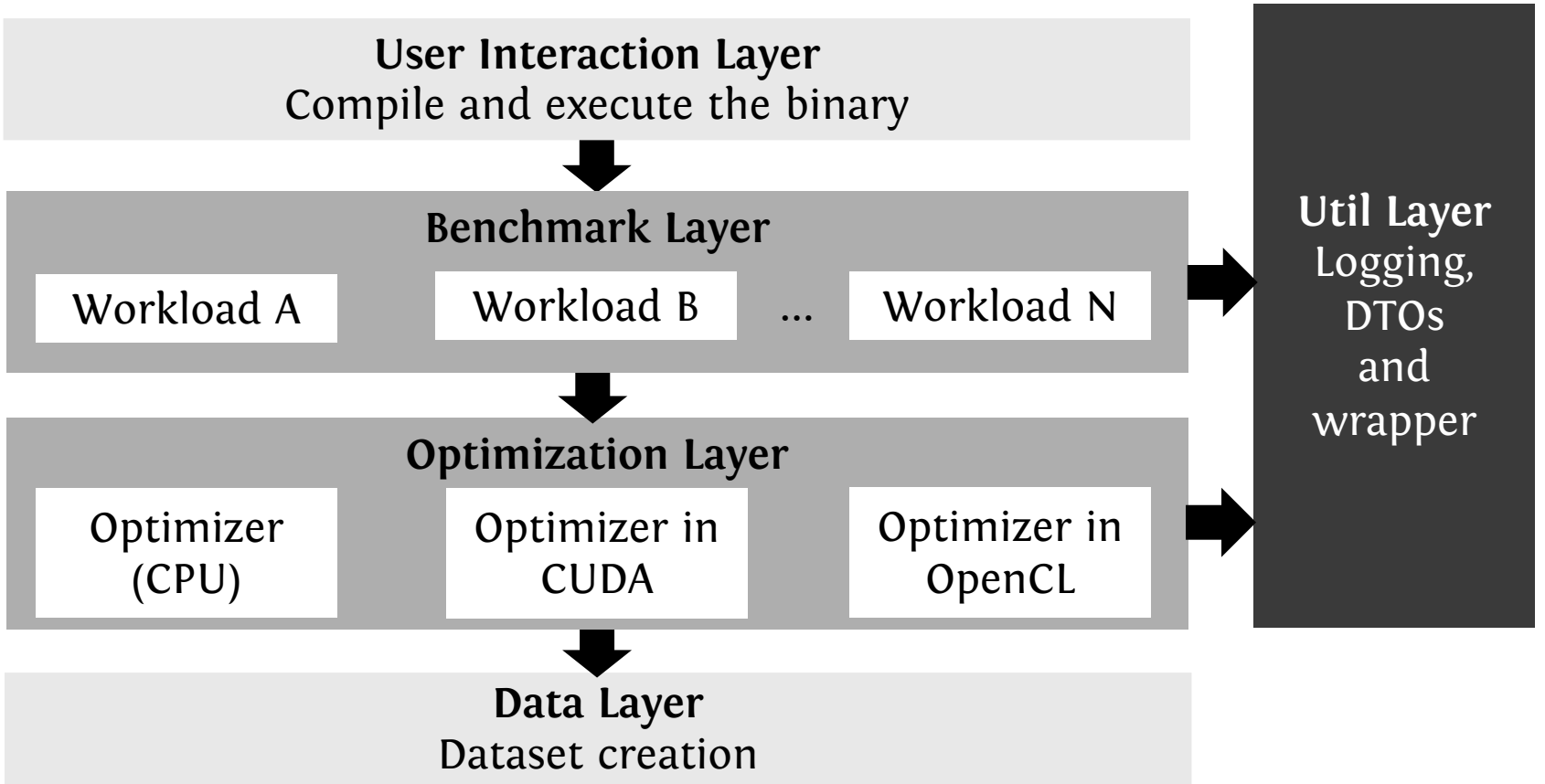
Important differences summarized…

- **Abstraction** does not expose hardware concepts.
- **Architecture** that's build for explicit control.
- **OpenCL C** is separated from host code

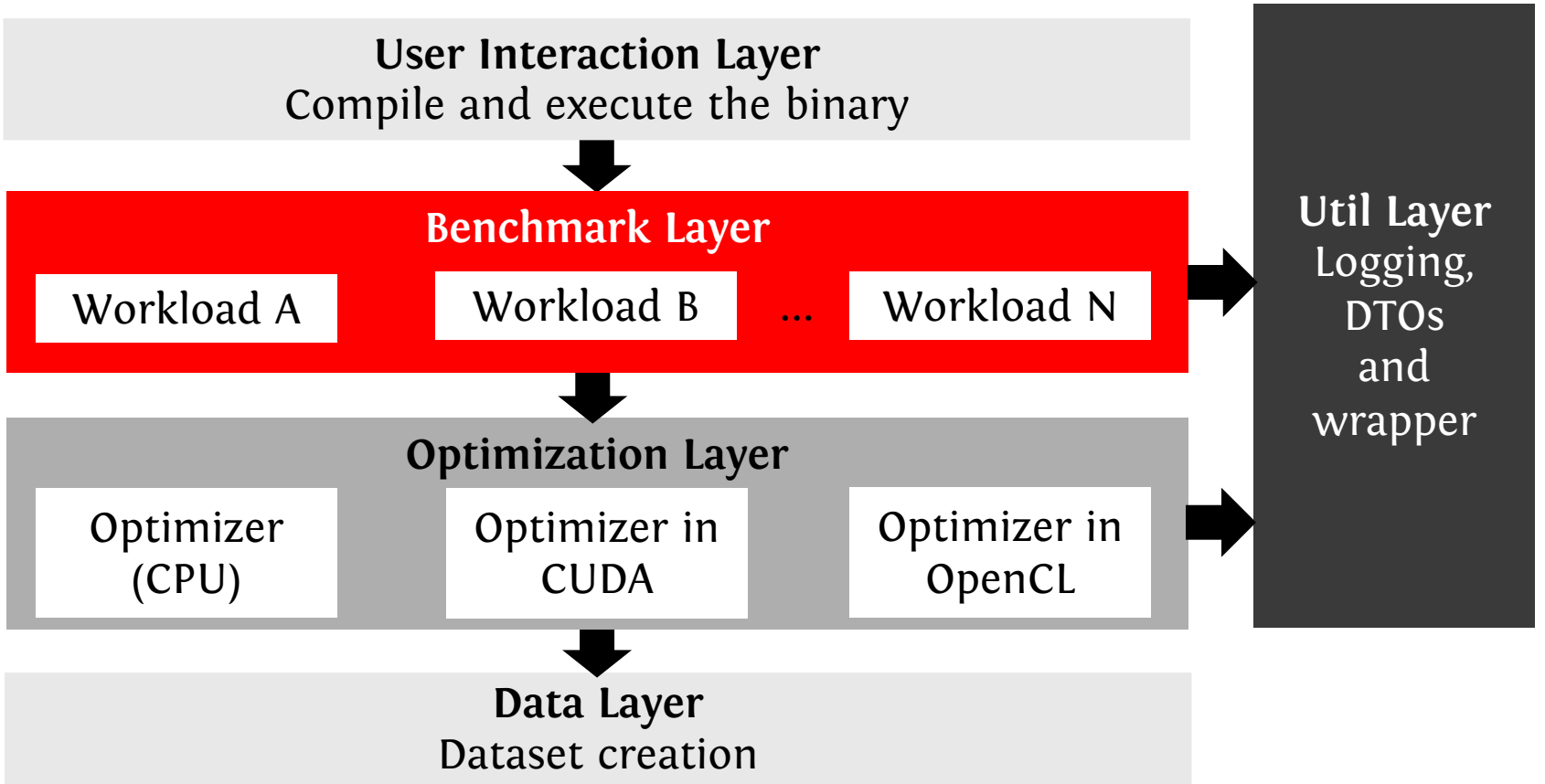
- **Abstraction** exposes hierarchical execution structure.
- **Architecture** simplifies interaction with runtime API.
- **CUDA C** is integrated within host code

The implementation

Architecture overview



Architecture overview



Benchmark layer

Definition and initialization of workloads...

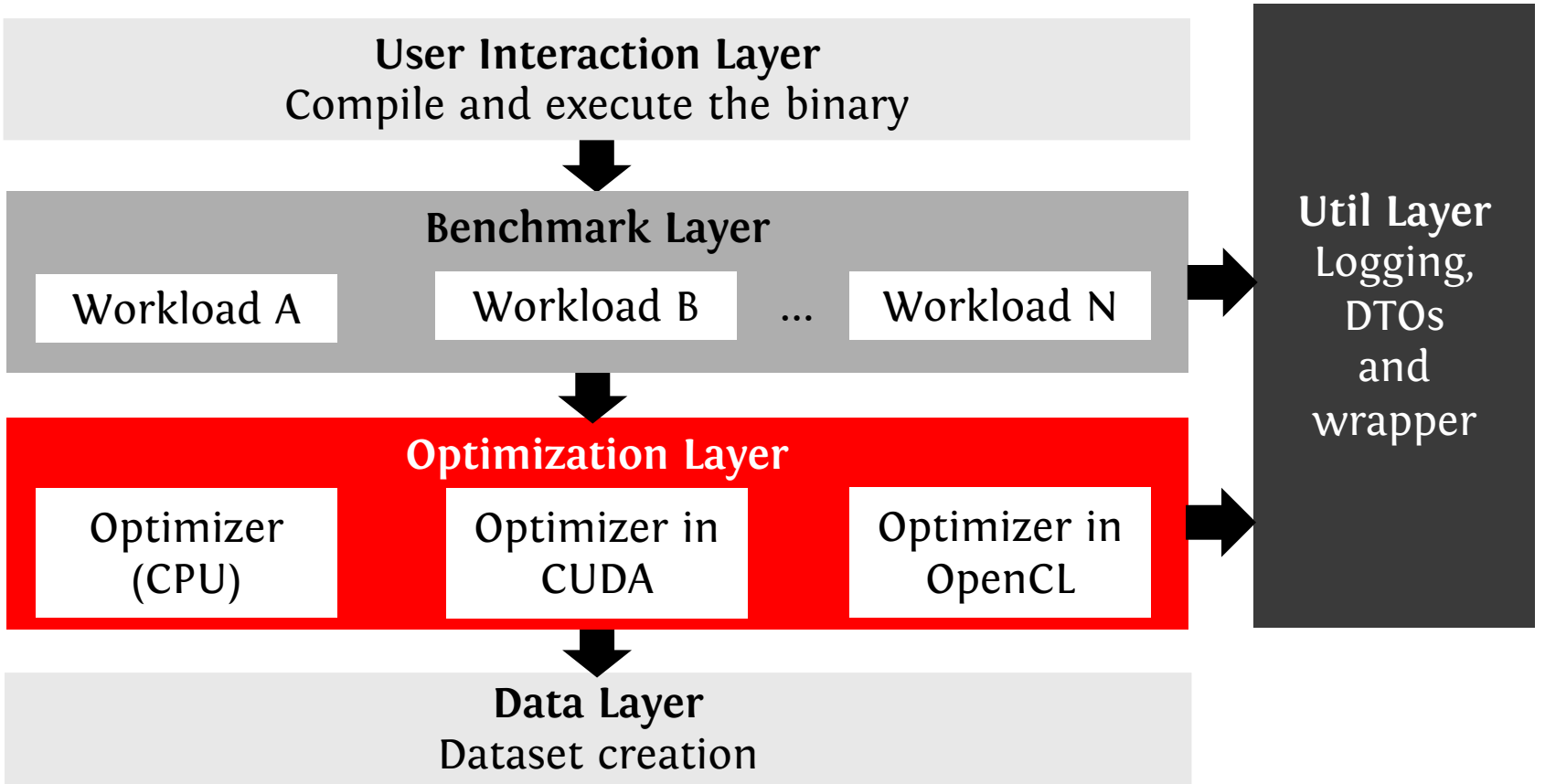
Benchmark Module
Define the workload and run with optimizer

Workload Modules

`<<Interface>>`
`Workload`
`forward(…) and compute_loss(…)`

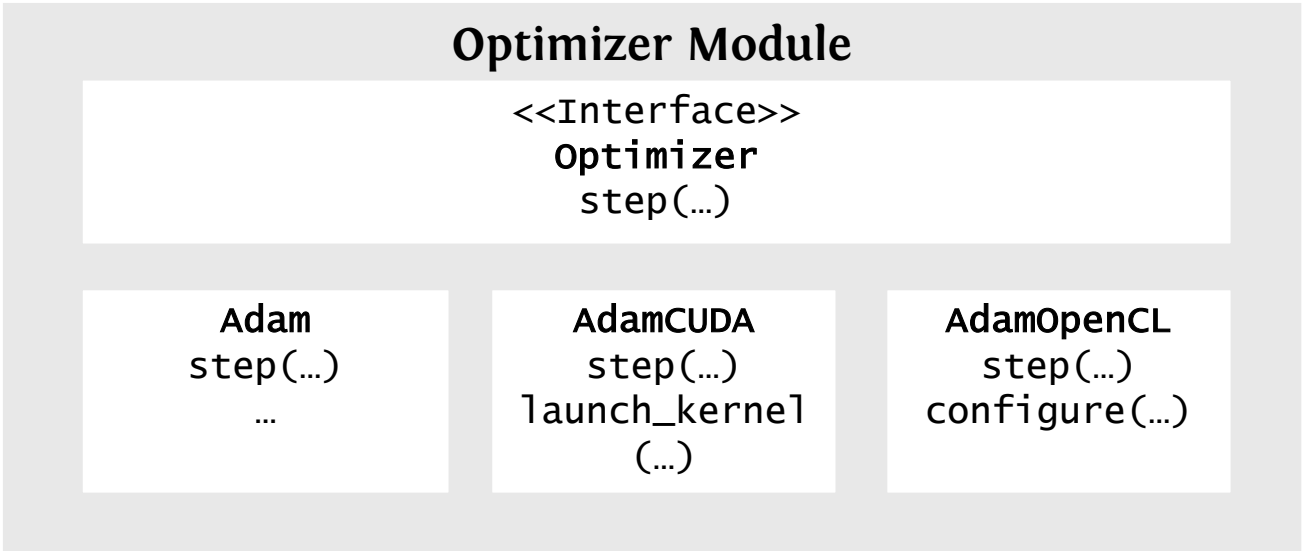
GEMM | **...**

Architecture overview

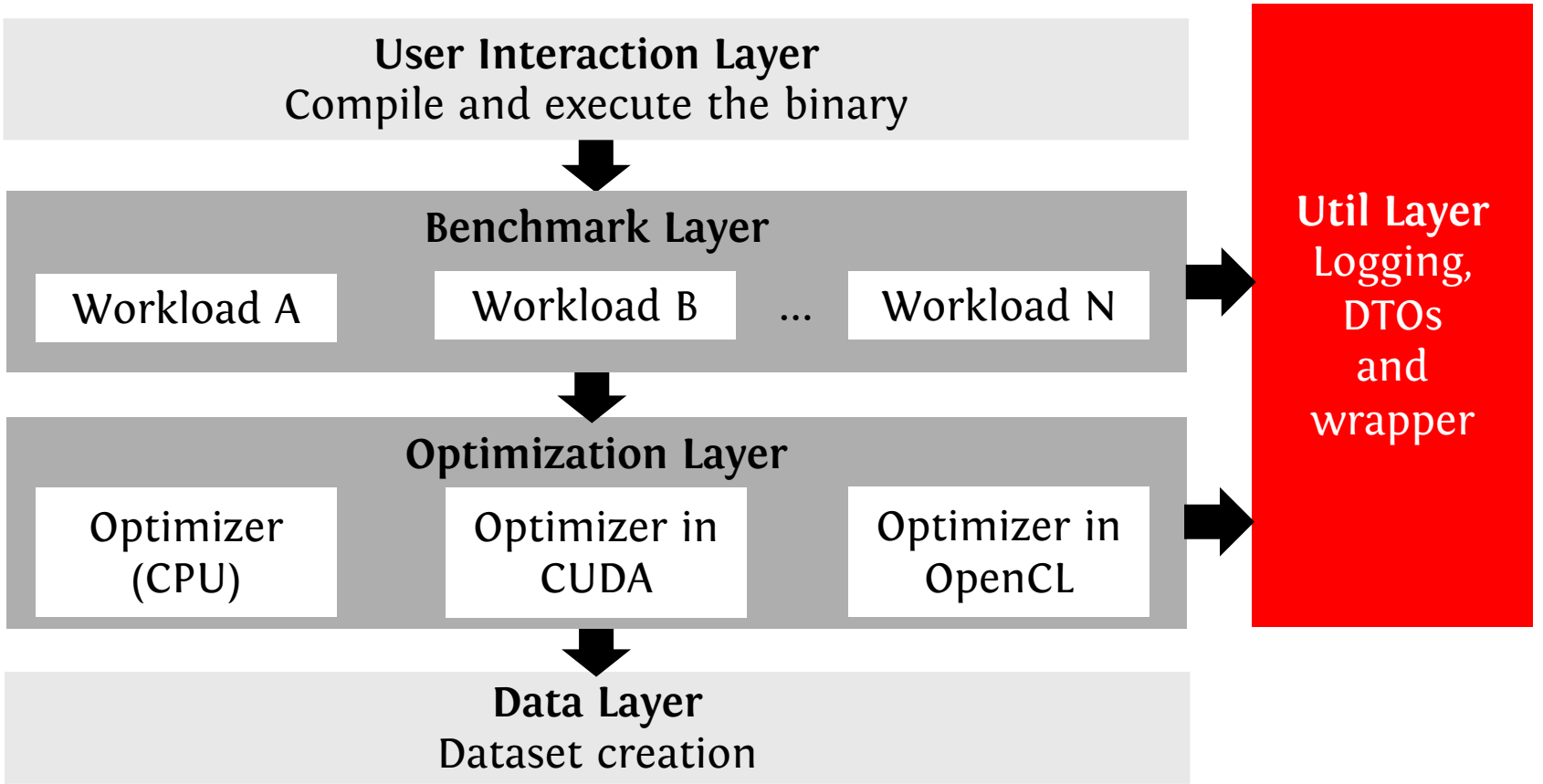


Optimization layer

Optimizer implementations on different platforms

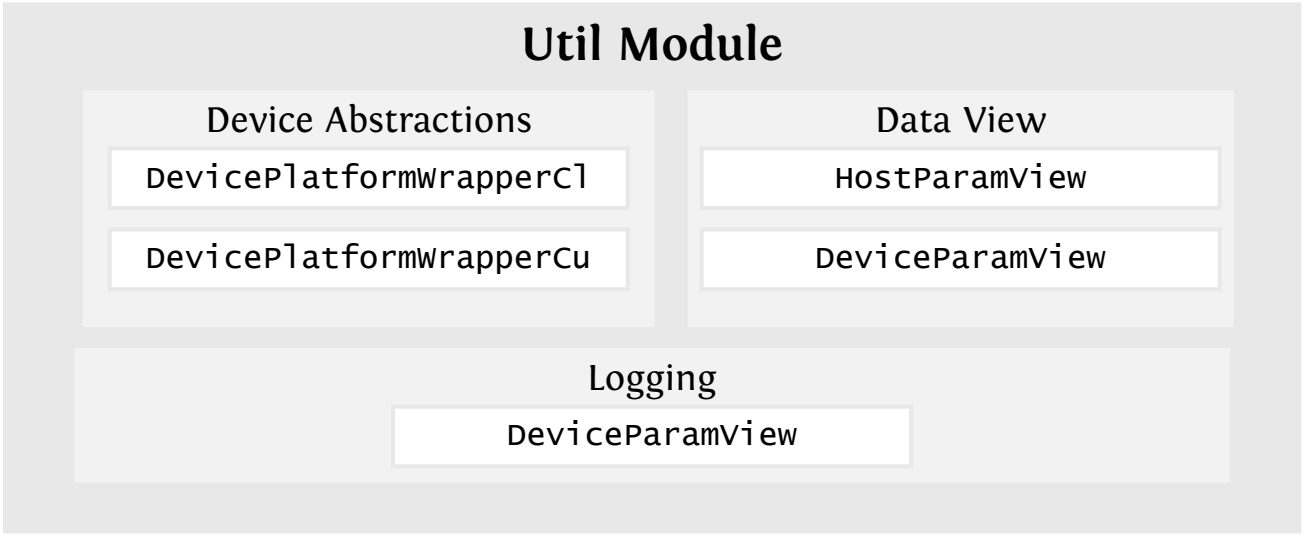


Architecture overview



`Util` layer

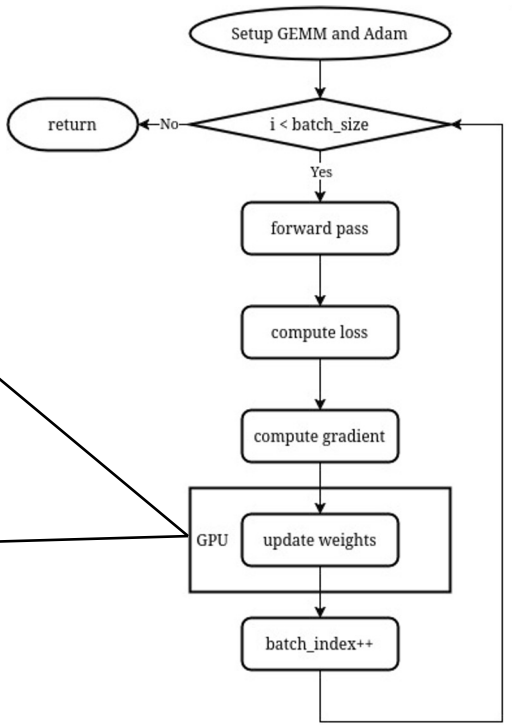
Implementation of cross-Cutting concerns



Results

Benchmark environment

```
__global__ void adam_update_kernel_impl(
        float *__restrict__ param,
        const float *__restrict__ grad,
        float *__restrict__ m,
        float *__restrict__ v,
        size_t n,
        float lr, float beta1,
        float beta2,
        float bc1, float bc2, float eps)
    {
        ...
    }
```



Measured aspects in consideration

Metric	Measurement
Computation-Time	**Isolate computer efficiency** of the programming model.
Data-Transfer-time	Workloads can be memory-bound instead of **compute-bound**.
Quality-of-solution	Provide **numerical stability** and correctness
Build-time	Insights into **compilation latency**.

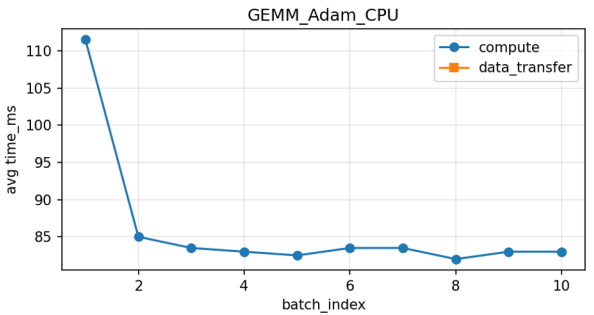
Benchmark environment

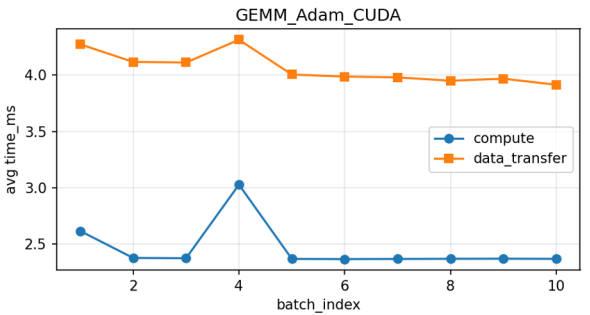
Running on…

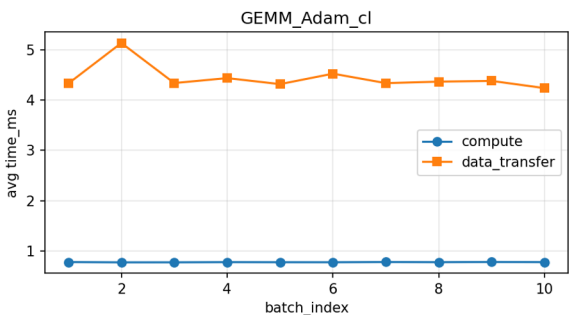
A NVIDIA GeForce GTX 970
with an Intel i3-9100F and 16 GB RAM

Visuals of currently obtained measurements

Averaged computation time and
data transfer time on batch-index....







Conclusion so far

OpenCL and CUDA perform on average pretty similar...

...but some Remarks:

- Currently only tested on GTX 970 → **old platform**
- The number of results are **sparse**
- Some metrics are missing, e.g. **build-time**

Outlook

Some TODOs until end of march...

- ✗ Platform independence
- ✗ AdamW
- ✗ More measurements

Sources

[1] Rishi, Rahul. (2011). Improving the Character Recognition Efficiency of Feed Forward BP Neural Network. International Journal of Computer Science & Information Technology. 3. 10.5121/ijcsit.2011.3107.
[2] Chi, Yujie & Schubert, Keith & Badal, Andreu & Roncali, Emilie. (2025). Review of GPU-based Monte Carlo simulation platforms for transmission and emission tomography in medicine. Physics in Medicine & Biology. 70. 10.1088/1361-6560/adfda7.
[3] Intel, "Intel Core i9 Prozessor 14900KF", accessed on 05 Feb 2026 from "https://www.intel.de/content/www/de/de/products/sku/236787/intel-core-i9-processor-14900kf-36m-cache-up-to-6-00-ghz/specifications.html"
[4] Nvidia, "Compare GeForce Graphics Cards" accessd on 05 Feb 2026 from https://www.nvidia.com/en-us/geforce/graphics-cards/compare/#50-series
[5] „From CUDA to OpenCL: Towards a Performance-portable Solution for Multi-platform GPU Programming" Peng D., Luszczek P., Tomov S., Peterson G., Dongarra J. Parallel Computing, Volume 38, Issue 8, Pages 391-407, 2012
[6] J. Fang, A. L. Varbanescu and H. Sips, "A Comprehensive Performance Comparison of CUDA and OpenCL," 2011 International Conference on Parallel Processing, Taipei, Taiwan, 2012, pp. 216-225, doi: 10.1109/ICPP.2011.45.
[7] Karimi, Kamran. "A Performance Comparison of CUDA and OpenCL." Arxiv Preprint ArXiv:1005.2581, arxiv.org, 2010.
[8] Holm, Havard Heitlo & Brodtkorb, André & Sætra, Martin. (2020). Performance and Energy Efficiency of CUDA and OpenCL for GPU Computing Using Python. 10.3233/APC200089.
[9] Puya Memarzia, Farshad Khunjush „An In-depth Study on the Performance Impact of CUDA, OpenCL, and PTX Code" Journal of Information and Computing Science Vol. 10, No. 2, 2015, pp.124-136
[10] Aftab M. et al. "OpenCL Programming Guide"2012 Pearson Education, Inc. ISBN-13: 978-0-321-74964-2
[11] Kirk D. et al. "Programming Massively Parallel Processors" 2010 David B. Kirk/NVIDIA Corporation and Wen-mei Hwu. Published by Elsevier Inc. ISBN: 978-0-12-381472-2
[12] NVIDIA "CUDA Programming Guide" 2026 accessed on 11.02.2026 from https://docs.nvidia.com/cuda/cuda-programming-guide/index.html