¡title¿

CONTENTS

I **Introduction** 2

II **Related Work** 3

III **The Implementation** 3

IV **Performance Benchmarks** 3

V **Conculion** 3

VI **References** 3

I. Introduction

Training a neural network typically requires large-scale data sets, such as ImageNet [?] or MNIST [?], to achieve robust generalization performance. As training involves multiple passes over these data sets and computationally expensive gradient-based optimization, the overall process is highly data-intensive.

To handle data-intensive workloads, data parallelism is required. Data parallelism refers to applying the same arithmetic operations to multiple data elements simultaneously. While modern CPUs support limited forms of data parallelism, their architecture is primarily optimized for low-latency and control-intensive tasks rather than high-throughput parallel computation [?]. GPUs, in contrast, are designed according to a Single-Instruction-Multiple-Data (SIMD) execution model, which enables a much higher degree of data parallelism. Therefore, utilizing GPUs for training neural networks is crucial for the efficient implementation of AI products and solutions. Various hardware vendors provide different software solutions to exploit GPU capabilities.

This work compares the performance of Graphics Processing Unit (GPU) frameworks such as OpenCL [?] and CUDA [?], which enable users to exploit the computational capabilities of GPUs. More precise it compares the solutions based on the benchmark defined in section **??**. To facilitate this comparison, various optimization algorithms are implemented using each framework. This approach is intended to provide insights into the performance characteristics of the different frameworks. In the presented application [?], several optimizers commonly used for training machine learning models are implemented, including Adam [?], AdamW [?], and Stochastic Gradient Descent [?].

CUDA is a software framework for utilizing GPUs, but it is restricted to hardware from the vendor NVIDIA. Furthermore, CUDA is neither standardized nor cross-platform. Nevertheless, it is the most commonly used solution for accessing GPUs in neural network training, as it is implemented in frameworks such as PyTorch, Keras, and TensorFlow [?]. This results in a strong dependence on NVIDIA graphics cards, as other vendors, such as AMD or ARM, are not compatible with the CUDA API [?]. In contrast, OpenCL provides a standardized, cross-platform parallel computing API based on C and C++. OpenCL is open source and maintained by the Khronos Group [?]. Therefore, this report aims to present the performance of non-NVIDIA-dependent solutions in comparison to NVIDIA-specific solutions. However, the application is executed on an NVIDIA GPU.

The different optimizer implementations are integrated into a Deep Convolutional Generative Adversarial Network (DCGAN), whose architecture is based on the PyTorch API Guide [?]. The model architecture, training procedure, and hyperparameters are kept identical across all experiments, with only the optimizer implementation differing between the CUDA- and OpenCL-based approaches.

II. Related Work

Several prior studies have compared OpenCL and CUDA. This section provides an overview of the considered comparison aspects, the evaluated workloads, and the main findings reported in related work.

Du P. et al. [?], compare CUDA and OpenCL are compared with respect to syntax, cross-platform compatibility, and overall computation and data transfer time. The evaluation is based on triangular solver (TRSM) and matrix multiplication (GEMM) workloads, with a particular focus on OpenCL performance across different platforms, including NVIDIA and ATI devices. The authors highlight OpenCL's functional portability, but demonstrate that performance portability is often limited by low-level architectural details. They attribute these limitations to differences in memory hierarchies, compiler optimizations, and architectural execution models.

Furthermore, Fang J. et al. [?], are evaluating performance is evaluated using a tailored performance ratio, defined as the ratio between the performance achieved by OpenCL and that achieved by CUDA. The study measures both device memory bandwidth and floating-point performance. In addition, similar to [?], the authors provide a conceptual comparison of the two frameworks, focusing on differences in terminology and programming abstractions used by CUDA and OpenCL. Performance is evaluated on 16 real-world workloads, including graph traversal, matrix transposition, and sparse matrix–vector multiplication. The experiments are conducted on multiple platforms from different vendors, such as NVIDIA and AMD. The results show that OpenCL's portability can lead to performance degradation in certain scenarios. One identified reason is that, on CPUs, OpenCL memory objects are implicitly cached by the hardware, making explicit use of local memory unnecessary and potentially harmful due to the introduced overhead.

In another comparision from Karimi K. et al. [?] OpenCL and CUDA are compared based on data transfer times to and from the GPU, kernel execution times, and end-to-end application execution times for both CUDA and OpenCL. Similarly as the others they also compare CUDA and OpenCL conceptually, in terms of code changes which are needed to transfer from CUDA to OpenCL. Here they describe the transfer was done with small work. As a workload they consider Adiabatic QUantum Algorthm, which is a monte carlo simulation of a quantum spin system written in C++. For they benchmark they state that choosing CUDA might be the better choise whenever performance is really important. Further the choose of CUDA and OpenCL depends on which hardware is available at the client side and what tools are there, as they state.

[?], [?], [?] or [?]

III. The Implementation

IV. Performance Benchmarks

V. Conculion

VI. References