Accelerating Machine Learning Inference on GPUs with SYCL

Ioanna-Maria Panagou
ipanagou@uth.gr
University of Thessaly
Volos, Greece

Nikolaos Bellas
nbellas@uth.gr
University of Thessaly
Volos, Greece

Lorenzo Moneta
Lorenzo.Moneta@cern.ch
CERN
Geneva, Switzerland

Sanjiban Sengupta
sanjiban.sengupta@cern.ch
CERN
Geneva, Switzerland

ABSTRACT

Recently, machine learning has established itself as a valuable tool for researchers to analyze their data and draw conclusions in various scientific fields, such as High Energy Physics (HEP). Commonly used machine learning libraries, such as Keras and PyTorch, might provide functionality for inference, but they only support their own models and are constrained by heavy dependencies, which render their deployment on embedded or bare-metal environments infeasible. SOFIE [3], which stands for **S**ystem for **O**ptimized **F**ast **I**nference code **E**mit, a part of the ROOT project developed at CERN, creates standalone C++ inference code from an input model in one of the popular machine learning formats. This code is directly invokable from other C++ projects and has minimal dependencies. In this work, we extend the functionality of SOFIE to generate SYCL code for machine learning model inference that can run on various GPU platforms and is only dependent on Intel MKL BLAS and portBLAS libraries.

CCS CONCEPTS

• **Computing methodologies → Parallel algorithms**; **Machine learning**.

KEYWORDS

Parallel Computing, SYCL, Machine Learning, Deep Learning, GPU

ACM Reference Format:
Ioanna-Maria Panagou, Nikolaos Bellas, Lorenzo Moneta, and Sanjiban Sengupta. 2024. Accelerating Machine Learning Inference on GPUs with SYCL. In *International Workshop on OpenCL and SYCL (IWOCL '24), April 8–11, 2024, Chicago, IL, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3648115.3648123

1 INTRODUCTION

ROOT [2] is an open-source data analysis framework, originating at CERN, used for high-scale data processing and analysis. ROOT offers native support for deep neural networks through the TMVA ROOT library [4]. SOFIE [3], an extension of the TMVA module, takes as input a trained PyTorch, Keras or ONNX model and creates standalone C++ inference code, which is directly invokable from other C++ projects and has minimal dependencies; only on BLAS libraries. In addition, it allows full control over the inference code, as well as seamless integration with other C++ projects.

Our objective is to demonstrate the generation of portable SYCL code for GPU inference with minimal dependencies, leveraging the SOFIE framework. Our contributions can be summarized as follows: (1) We enhanced the SOFIE framework, enabling the generation of SYCL code tailored for efficient machine learning model inference, relying only on Intel MKL BLAS and portBLAS libraries and targeting a variety of GPU platforms (Intel, NVIDIA and AMD) (2) We included practical insights and optimization tips for SYCL code specifically for machine learning inference tasks and (3) We conducted thorough benchmarking on diverse machine learning models across different platforms and compared our implementation against CPU execution and ONNXRUNTIME.

2 BACKGROUND

SOFIE accepts input in the form of a pre-trained machine learning model, presented in ONNX (`.onnx`), PyTorch (`.pt`), or Keras (`.h5`) format. A parser transforms the input model into an instance of the `SOFIE::RModel` class, which is essentially a graph representation comprised of nodes that correspond to operators, such as GEMM (GEneral Matrix Multiplication) or CONV (Convolution).

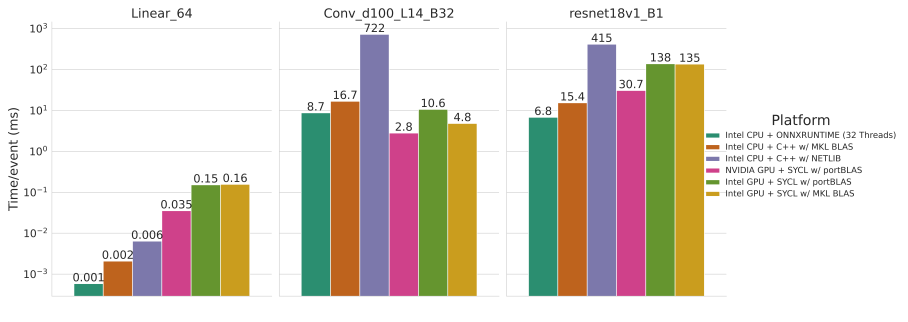
Following the construction of the `RModel`, the code generation process is initiated. This step produces 2 outputs: a weight file in `.dat` or `.root` data format and a C++ header file (`.hxx`) that hardcodes the inference function and can be included in any C++ project. The resulting code can be compiled as usual, but it has to be linked against the BLAS libraries.

3 IMPLEMENTATION

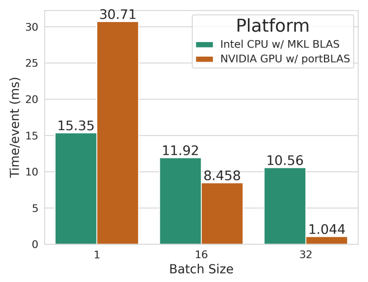
To ensure consistency and mitigate the risk of errors, we utilize the buffer/accessor model and delegate the choice of number of work-items in a work-group to the runtime.

3.1 Performance Considerations

3.1.1 Efficient Data Movement To minimize the overhead associated with data transfers between the host and the device, we have implemented all layers on the GPU and transfer the data only once at the beginning from host to device and once at the end of computation in the opposite direction. To avoid triggering copies back to the host in-between layers, we set the accessor mode for the output of each layer to `write_only` and manage buffers efficiently



(a)



(b)

Figure 1: Left: Time/event in ms for different benchmarks per Platform and BLAS library (Logarithmic Scale) Right: Time/event in ms vs Batch Size for resnet18_v1. The NVIDIA GPU is an NVIDIA GeForce RTX 4090 and the Intel GPU is an Intel Arctic Sound-P.

by declaring them at the start of the scope and closing the latter after all kernel launches. For input and intermediate tensors, we specify `nullptr` as the final data destination to ensure that their data will not be unnecessarily copied back to the host. Furthermore, we use `sycl::no_init` for layer output accessors to inform the runtime that previous buffer contents can be discarded. Finally, the `use_host_ptr` property during buffer creation enables direct use of host memory by the buffer if possible.

3.1.2 Using Libraries for GPU Offloading For routines such as `gemm`, `axpy` and `scal` we utilized the respective BLAS routines offered by the oneAPI MKL BLAS library (only for Intel GPUs) and the portBLAS library.

3.1.3 Reduction In parallel programming, reduction applies an operator to all array elements, yielding a single result. SYCL 2020 introduced reduction variables, eliminating the need for manual handling of parallel reductions.

3.1.4 Kernel Fusion For some subsequent operations, like clipping and activation, SOFIE C++ generated code used two subsequent loops. We fused the two subsequent loops into one kernel launch, potentially reducing execution time.

3.1.5 Replacing Conditional Checks with Relational Functions To mitigate the issue of thread divergence, we replaced conditional checks with relational functions wherever possible, to ensure that work-items do not execute different paths.

3.2 Convolution/Deconvolution

For the convolution and deconvolution operators, we employed the established im2col [1] approach. During runtime, we dynamically convert the input image and weights into 2D arrays and treat the convolution as a matrix multiplication, using BLAS library routines.

4 EXPERIMENTAL EVALUATION

Although the presented benchmarks pertain to HEP experiments, we are able to generate SYCL code for any machine learning model. The metric we use for evaluation is `time/event`. In the context of HEP, an event is the set of data points collected by a particle accelerator for each collision of accelerated particles.

We tested 3 different GPU configurations: Intel GPU using MKL BLAS, Intel GPU using portBLAS, and NVIDIA GPU using portBLAS. Figure 1a shows the time per event in milliseconds for a representative sample of benchmarks for each of those configurations and also compares them with the execution time on a 13th Gen Intel(R) Core(TM) i9-13900KF with 32 threads operating at (maximum) 5.8GHz using plain C++ with BLAS libraries and ONNXRUNTIME.

The MKL BLAS library outperforms portBLAS for Intel GPU devices, and it surpasses NETLIB libraries for CPUs due to its multithreaded nature. Notably, our SYCL GPU implementation achieves significant speedups; x258 over the C++ implementation and x3.12 over multi-threaded ONNXRUNTIME inference for the CONV_d100_L14_B32 model. However, for small models like Linear_64, GPU performance is comparatively worse, suggesting that the model size may not fully leverage GPU resources.

Model size also depends on the input batch size. Figure 1b illustrates the execution time per event in ms for the resnet_v1 model with varying batch sizes. Increasing batch size enhances parallelism, enabling the GPU to outperform the parallel CPU implementation.

5 CONCLUSIONS AND FUTURE WORK

Our work lays the foundation for the adoption of SYCL for machine learning model inference outside the field of HEP. The efficiency of our framework is more evident with larger models, aligning with the prevailing trend in the field. However, there is still room for improvement if we wish to surpass the performance of existing frameworks including using the USM model for fine-grained control over data transfers and leveraging the Intel DNN libraries for deep learning building blocks.

REFERENCES

[1] Andrew Anderson et al. 2020. High-performance low-memory lowering: GEMM-based algorithms for DNN convolution. In *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 99–106.
[2] M. Ballintijn I. Antcheva et al. 2011. ROOT — A C++ framework for petabyte data storage, statistical analysis and visualization. *Computer Physics Communications* 182, 6 (2011), 1384–1385. https://doi.org/10.1016/j.cpc.2011.02.008
[3] Lorenzo Moneta Sitong An et al. 2023. SOFIE: C++ Code Generation for Fast Inference of Deep Learning Models in ROOT/TMVA. *Journal of Physics: Conference Series* 2438, 1 (feb 2023), 012013. https://doi.org/10.1088/1742-6596/2438/1/012013
[4] Jan Therhaag. 2010. TMVA Toolkit for multivariate data analysis in ROOT. *PoS* ICHEP2010 (2010), 510. https://doi.org/10.22323/1.120.0510