Towards performance portability of AI models using SYCL-DNN

Muhammad Tanvir
muhammad.tanvir@codeplay.com
Codeplay Software Ltd.
Edinburgh, Scotland, UK

Kumudha Narasiman
kumudha.narasimhan@codeplay.com
Codeplay Software Ltd.
Edinburgh, Scotland, UK

Mehdi Goli
mehdi.goli@codeplay.com
Codeplay Software Ltd.
Edinburgh, Scotland, UK

Ouadie El Farouki
ouadie.elfarouki@codeplay.com
Codeplay Software Ltd.
Edinburgh, Scotland, UK

Svetlozar Georgiev
svetlozar.georgiev@codeplay.com
Codeplay Software Ltd.
Edinburgh, Scotland, UK

Isaac Ault
isaac.ault@codeplay.com
Codeplay Software Ltd.
Edinburgh, Scotland, UK

ABSTRACT

The wide adoption of Deep Neural Networks *(DNN)* has served as an incentive to design and manufacture powerful and specialized hardware technologies, targeting systems from Edge devices to Cloud and supercomputers. This huge diversity soon becomes a burden due to the emerging dependencies between development stacks and deployment hardware.

While the proposed *ONNX* as a de facto for AI model description, provides the portability of AI models across various AI frameworks, supporting DNN models on various hardware architectures remains challenging. Several existing AI frameworks such as Tensorflow, Pytorch, ONNXRuntime provides performance portability via a dedicated backend implementations per hardware architecture. While such approach provides wider support of hardware devices, maintainability and readability remains challenging.

There are many libraries and frameworks which were developed to support neural network models and we discuss some of the important ones in this section.

Frameworks like Glow [18], nGraph [14] and Tensor Comprehensions [19] use a compiler-based approach to accept the neural network model and emit optimised code for a specific hardware.

The neural network model is lowered into one or more intermediate representations before generating an optimised kernel. These frameworks, target a specific set of backends and targeting any new hardware requires implementing a considerable fraction of the operators. Other frameworks like Caffe [16], PyTorch [17] and TinyNN [10] provide runtime solution by integrating various vendor specific libraries or graph as a backend to support neural network models on different set of architectures. Framework like TensorFlow [11], rely on calling vendor-specific libraries or graph compilers. While embedding vendor-specific library can lead to achieving near metal performance, it can make adding and maintaining different backends quite tedious.

Intel oneMKL [4] and oneDNN [7] are the optimized libraries for linear algebra subroutine and deep neural network routines for multi-core and manycore Intel systems. Recently, oneMKL and oneDNN have added support for running on Nvidia GPUs as well [15] via SYCL interoperability with third party libraries. This approach integrates the existing vendor optimised backend in SYCL to provide a unique SYCL-interface for memory management and runtime control from the user point of view while reusing the highly optimised vendor backend.

ARM Compute Library [1], cuBLAS [6] and cuDNN [13] , MIOpen [5] provides optimised routines for linear algebra and machine learning for ARM, Nvidia and AMD respectively. All these libraries are optimised for specific architectures, and very rarely provide portability.

IWOCL'22, May 10–12, 2022, Bristol, United Kingdom, United Kingdom
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9658-5/22/05.
https://doi.org/10.1145/3529538.3529999

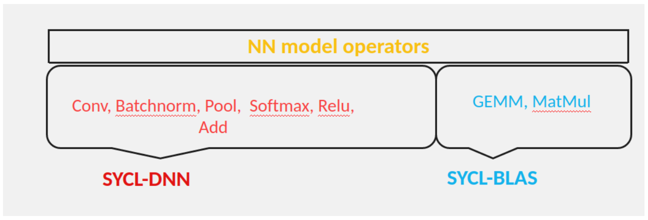

Figure 1: Neural network operator mapping in SYCL-DNN and SYCL-BLAS

SYCL provides a C++-based portable parallel programming model to target various devices like CPUs, GPUs, DSPs, FPGAs, etc. SYCL programming model allows the developers to write highly parametrized kernels for a diverse hardware set in a unified setting. These kernels can then be tuned for the specified hardware. Hence, enabling *SYCL* backend for an AI framework (e.g. Tensorflow, Pytorch etc.) can lead to a hardware agnostic model for heterogeneous systems and also allow to reuse the existing optimized library implementations.

Libraries like SYCL-BLAS [8] and SYCL-DNN [9] are open source and are a part of the SYCL eco-system. They can be compiled with any SYCL compiler such as ComputeCPP [2] or DPC++ [3] and run on any SYCL-enabled devices. ComputeCPP also supports SYCL RISC-V architecture [12]. Thus, making the applications using these libraries sufficiently portable.

The SYCL kernels implemented in SYCL-DNN and SYCL-BLAS allow for tuning parameters such as cache size, work group size and local memory size based on the hardware we execute on. This helps reuse the existing kernels but still provide good performance on new hardware by tuning for these finer details.

Intel CPU			Intel GPU			Nvidia GPU		
SYCL-DNN	SYCL-DNN + tuning	oneDNN	SYCL-DNN	SYCL-DNN + tuning	oneDNN	SYCL-DNN	SYCL-DNN + tuning	cuDNN
286	174	240	151.2	125.2	103	13.98	7.33	6.6

Table 1: Execution time of VGG-16 network (in ms) with SYCL-DNN, oneDNN and cuDNN libraries

SYCL-DNN already supports OpenCL backend and in this paper we extend SYCL-DNN to support Nvidia and RISC-V architectures. Figure 1 shows the NN operation mapping. The results provide a detailed analysis of the performance portability of SYCL based AI frameworks on various architectures with respect to state-of-the-art optimized vendor specific libraries.

The performance of SYCL-DNN is in-par on existing OpenCL backends as compared to device specific optimized libraries. We run the VGG model to understand and compare performance (Table 1). In case of Intel GPU - HD Graphics 530, SYCL-DNN provides 80% the performance of optimized oneDNN execution provider. The performance gap in this case is because of the extra graph optimizations that oneDNN performs. For Intel CPU, we used ComputeCPP 2.4 as the SYCL compiler and the latest oneDNN from github repository. We observe that SYCL-DNN performs 19% slower than oneDNN. However, tuning the matmul operation in SYCL-DNN provides a considerable speedup and SYCL-DNN performance 37% better than oneDNN.

We extend SYCL-DNN to support DPC++ as one of the SYCL compilers. DPC++ provides support for CUDA backend, there by enabling running SYCL kernels on Nvidia devices. We compare the performance with optimized cuDNN libraries. We used the latest DPC++ as the SYCL compiler and cuDNN version 7.6.5. We see that untuned SYCL-DNN is almost 50% slower than cuDNN. This is because the matmul implementation in SYCL-DNN does not optimize for local memory. Further tuning and using the optimized SYCL-BLAS implementation of matmul improves the performance and we observe that SYCL-DNN comes within 90% of the performance of cuDNN. cuDNN has hand-written optimized implementation of some of the routines and hence achieves 10% more performance then SYC-DNN but, the code written in cuDNN cannot be reused on any other hardware.

Further, there are no Execution providers / frameworks which provide full support for RISC-V architectures. By integrating SYCl-DNN with the Acoran compute stack, we are able to support generating RISC-V ISA. The Acoran compute stack uses ComputeCPP and ComputeAorta to enable running SYCL Kernels on RISC-V architectures. We run the VGG-16 model on the RISC-V Spike simulator. The current implementation of the simulator is single core and hence VGG-16 takes 312 seconds and ResNet-50 takes 198 seconds to complete execution. In case of VGG-16, the simulator requires 16532358513 cycles to finish execution.

We are enabling SYCL backend for ONNXRuntime as a future work to exploit the benefit of ONXX model loader to load the ONXX model from different AI framework and also benefit from ONXX runtime graph optimisation.

ACM Reference Format:
Muhammad Tanvir, Kumudha Narasiman, Mehdi Goli, Ouadie El Farouki,, Svetlozar Georgiev, and Isaac Ault. 2022. Towards performance portability of AI models using SYCL-DNN. In *International Workshop on OpenCL (IWOCL'22), May 10–12, 2022, Bristol, United Kingdom, United Kingdom.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3529538.3529999

REFERENCES

[1] Accessed: 2022-03-08. The ARM Computer Vision and Machine Learning library. https://github.com/ARM-software/ComputeLibrary/.
[2] Accessed: 2022-03-08. ComputeCPP compiler. https://developer.codeplay.com/products/computecpp/ce/home.
[3] Accessed: 2022-03-08. DPC++ compiler. https://github.com/intel/llvm.
[4] Accessed: 2022-03-08. Intel® Math Kernel Library. https://software.intel.com/content/www/us/en/develop/tools/math-kernel-library.html.
[5] Accessed: 2022-03-08. MIOpen: AMD's Machine Intelligence Library. https://github.com/ROCmSoftwarePlatform/MIOpen.
[6] Accessed: 2022-03-08. NVIDIA cuBLAS: Dense Linear Algebra on GPUs. https://developer.nvidia.com/cublas.
[7] Accessed: 2022-03-08. OneAPI Deep Neural Network Library (oneDNN). https://01.org/onednn.
[8] Accessed: 2022-03-08. SYCL-BLAS: An implementation of BLAS using the SYCL open standard. https://github.com/CodeplaySoftware/SYCL-BLAS.
[9] Accessed: 2022-03-08. The SYCL-DNN neural network acceleration library. https://github.com/CodeplaySoftware/SYCL-DNN.
[10] Accessed: 2022-03-08. TinyNN. https://github.com/borgwang/tinynn.
[11] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi
[12] Rod Burns, Colin Davidson, and Aidan Dodds. 2021. Enabling OpenCL and SYCL for RISC-V processors. In *International Workshop on OpenCL*. 1–1.
[13] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient Primitives for Deep Learning. arXiv:1410.0759 [cs.NE]
[14] Scott Cyphers, Arjun K. Bansal, Anahita Bhiwandiwalla, Jayaram Bobba, Matthew Brookhart, Avijit Chakraborty, Will Constable, Christian Convey, Leona Cook, Omar Kanawi, Robert Kimball, Jason Knight, Nikolay Korovaiko, Varun Kumar, Yixing Lao, Christopher R. Lishka, Jaikrishnan Menon, Jennifer Myers, Sandeep Aswath Narayana, Adam Procter, and Tristan J. Webb. 2018. Intel nGraph: An Intermediate Representation, Compiler, and Executor for Deep Learning. arXiv:1801.08058 [cs.DC]
[15] Mehdi Goli, Kumudha Narasimhan, Ruyman Reyes, Ben Tracy, Daniel Soutar, Svetlozar Georgiev, Evarist M Fomenko, and Eugene Chereshnev. 2020. Towards cross-platform performance portability of dnn models using sycl. In *2020 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. IEEE, 25–35.
[16] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia* (Orlando, Florida, USA) *(MM '14)*. Association for Computing Machinery, New York, NY, USA, 675–678. https://doi.org/10.1145/2647868.2654889
[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach,

H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8026–8037. http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[18] Nadav Rotem, Jordan Fix, Saleem Abdulrasool, Garret Catron, Summer Deng, Roman Dzhabarov, Nick Gibson, James Hegeman, Meghan Lele, Roman Levenstein, Jack Montgomery, Bert Maher, Satish Nadathur, Jakob Olesen, Jongsoo Park, Artem Rakhov, Misha Smelyanskiy, and Man Wang. 2018. Glow: Graph Lowering Compiler Techniques for Neural Networks. arXiv:1805.00907 [cs.PL]

[19] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S. Moses, Sven Verdoolaege, Andrew Adams, and Albert Cohen. 2018. Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions. arXiv:1802.04730 [cs.PL]