OpenCL Acceleration for TensorFlow

Mehdi Goli, Luke Iwanski, John Lawson, Uwe Dolinsky, and Andrew Richards
Codeplay Software Ltd.
Edinburgh, UK
{mehdi.goli,luke,john,uwe,andrew}@codeplay.com

ABSTRACT

There is huge demand for targeting complex and large-scale machine learning applications particularly those based on popular actively-maintained frameworks such as TensorFlow and CAFFE to a variety of platforms with accelerators ranging from high-end desktop GPUs to resource-constrained embedded or mobile GPUs, FPGAs, and DSPs. However, to deliver good performance different platforms may require different algorithms or data structures, yet code should be easily portable and reused as much as possible across different devices. The open SYCL standard addresses this by providing parallel processing through a single-source programming model enabling the same standard C++ code to be used on the CPU and accelerator. This allows high-level C++ abstractions and templates to be used to quickly configure device and host code to cover specific features of the platform. By targeting OpenCL, SYCL enables C++ applications such as TensorFlow to run efficiently on OpenCL devices without having to write OpenCL code.

1 INTRODUCTION

Machine learning has been widely used in different areas, such as image recognition and self-driving vehicles [1, 3, 10, 12].

There are several deep neural networks (DNN) software frameworks for constructing and executing DNN models such as CAFFE [13], MXNet [4], TensorFlow [1], Theano [2], and Torch [6]. TensorFlow is a data-flow graph model used for constructing DNN models for recognizing patterns. Each node in the graph represents a unit of computations called an *operator*. Each edge represents an input/output data called a *Tensor*. Each operator can have $k \geq 0$ specializations for different devices. A device-specialized operator can be directly implemented in the TensorFlow framework, or can be a wrapper around a third-party library implementing the actual unit of computation for that operator. The main third-party libraries in TensorFlow are Eigen [11] and CuDNN [5].

TensorFlow has some features which distinguishes it from other DNN frameworks. Being a graph-based model, TensorFlow supports graph-based optimization techniques such as optimized deployment and deferred execution model. The latter allows the framework to have global information about the graph in order to issue a sequence of GPU kernels to the device without waiting for intermediate results. Also, unlike CAFFE that provides a set of high-level layers for constructing a neural network model, TensorFlow provides a set of primitive operators as graph nodes. This gives more flexibility to users for expressing the model or adding a new layers or optimizers. Moreover, TensorFlow has an abstraction layer to support the possibility of adding any future devices to the framework. [1]. While some computations expressed in DNN models can be executed across heterogeneous systems, support has so far been limited to NVIDIA graphics processors using CUDA for almost all existing frameworks. Therefore, the above abstraction layer in TensorFlow plays a significant roll to ease up the feasibility of adding OpenCL support for other processors such as ARM, AMD, and Intel GPUs and FPGAs.

In this paper we propose an OpenCL-enabled back-end for TensorFlow via SYCL in order to enable developers to access a wider range of processor combinations.

SYCL [18] is a royalty-free, cross-platform C++ abstraction layer that builds on the underlying concepts, portability and efficiency of OpenCL, while adding the ease-of-use and flexibility of modern C++14. This solution also benefits from ensuring the implementation is maintainable and compliant as the standards evolve.

Dispatching device kernels from C++ applications is a widely used method for dealing with heterogeneous platforms in various programming models, such as CUDA [15], C++AMP [14], HCC [17], OpenACC [9], or OpenMP [16]. SYCL brings this capability to a wide range of accelerators supporting OpenCL.

This lets developers create powerful, more performance-portable template libraries that can take advantage of a wide range of heterogeneous hardware and software platforms. Moreover, porting TensorFlow to OpenCL would mean handwriting the kernels in OpenCL C and having separate code-bases, which would be complicated to maintain. By using SYCL, everything is single-source C++, and therefore it is possible to use a non-intrusive approach to add the SYCL back-end to TensorFlow.

2 THE APPROACH

There are three steps for implementing the SYCL back-end for TensorFlow: In the first step we have introduced the SYCL device by specializing the TensorFlow's device abstraction layer. The implemented SYCL device supports any OpenCL-enabled devices. In the next step we implement a SYCL back-end for all the linear operations in the Eigen Tensor module. The detailed implementation of a SYCL back-end for Eigen Tensor module can be found in [7, 8]. Each TensorFlow operator maps to an Eigen expression. As the Eigen has the same expression interface regardless of the selected device, in most cases, there is no need to specialize TensorFlow's operators for SYCL. In the last step we have registered the existing TensorFlow operators for SYCL device. Listing 1 represents the registration of the TensorFlow `sqrt` operation for SYCL, CPU and CUDA respectively.

Through SYCL, we have also provided the OpenCL interoperability mode. This will allow users to embed hand-tuned OpenCL code for expensive operations such as matrix multiplication or convolution.

Figure 1 represents the TensorFlow architecture for the SYCL back-end.

```
namespace tensorflow {
REGISTER5(UnaryOp, CPU, "Sqrt", functor::sqrt, float, Eigen::half, double,
          complex64, complex128);
#if GOOGLE_CUDA
REGISTER3(UnaryOp, GPU, "Sqrt", functor::sqrt, float, Eigen::half, double);
#endif
#ifdef TENSORFLOW_USE_SYCL
REGISTER2(UnaryOp, SYCL, "Sqrt", functor::sqrt, float, double);
#endif // TENSORFLOW_USE_SYCL
}
```

Listing 1: Registration of the `sqrt` operations in TensorFlow

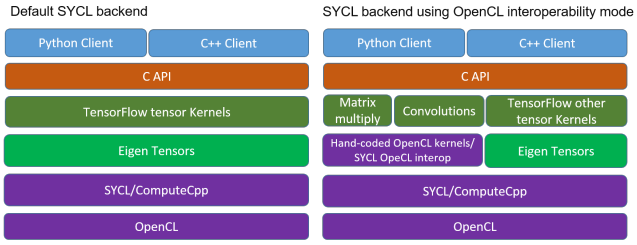

Figure 1: TensorFlow architecture for SYCL back-end

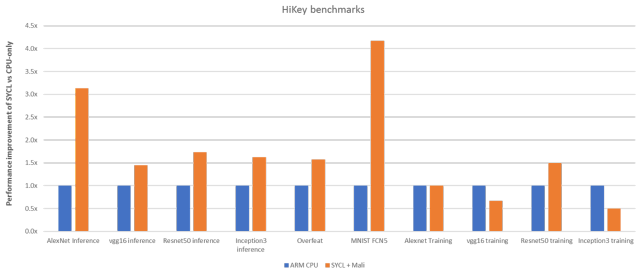

Figure 2: DNN model speed-up over A53 CPU on Hikey board

3 EVALUATION

The evaluation of the proposed approach was carried out on an ARM Hikey board with A53 CPU and Mali GPU.

Figure 2 represents the speed up we can achieve for DNN models over A53 ARM CPU when using SYCL back-end on ARM Mali. As shown in the figure 2, for the inference model we can achieve up to 4 times speed up over the A53 CPU back-end. For the training models the bottleneck is the back propagation algorithm proposed for SYCL. Currently we are working on improving the performance of the back propagation algorithm for convolutional neural networks.

As Eigen has been used as the backbone of the TensorFlow operator, we compare the performance of the Eigen implementation of TensorFlow operators registered for SYCL on Arm Mali in order to further analyze the SYCL kernels. We have divided the Eigen operations into compute-bound and memory-bound operations.

Figures 3 and 4 represent up to an order of magnitude speed-up over Arm A53 CPU for both memory-bound and compute-bound kernels. However, there are cases like tensor chipping and shuffling

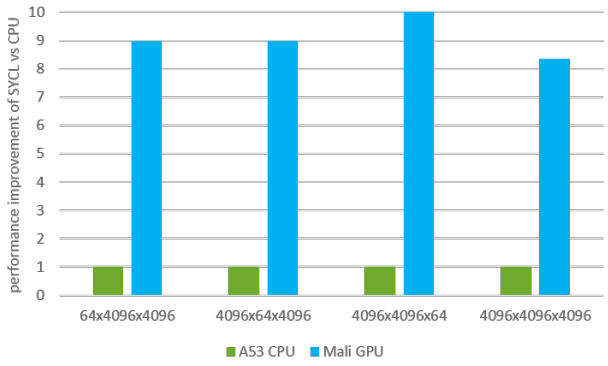

Figure 3: Tensor contraction speed-up over A53 CPU on Hikey board.

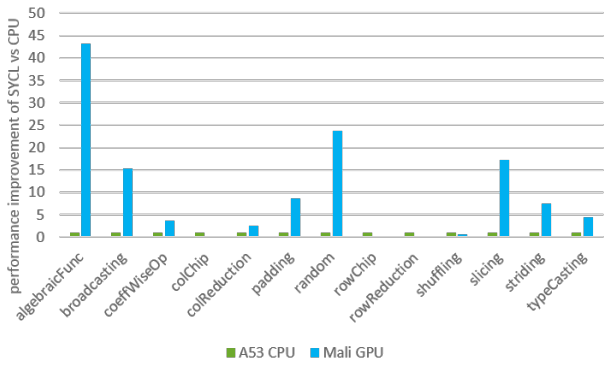

Figure 4: Eigen kernels speed-up over A53 CPU on Hikey board. The input data size is 4k.

operations where we can see a drop in performance. One reason for such behaviour could be due to non-coalesced memory access. Non-coalesced memory accesses may have more performance impact on GPU than CPU. Such an example can be chipping a row of a matrix where the data layout is column major. Also, the reduction algorithm is a naive implementation suffering from non-coalesced memory accesses and we are working on its improvement.

4 CONCLUSIONS

This paper represents an OpenCL back-end for TensorFlow using SYCL. Our results show significant improvement over those run on ARM A53 CPU, specially for large-scale applications. We are maintaining the TensorFlow SYCL back-end and actively optimizing TensorFlow's operators for different DNN models across different platforms.

REFERENCES

[1] Martín Abadi et al. 2016. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA.*

[2] Rami Al-Rfou et al. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688* (2016).

[3] Anelia Angelova, Alex Krizhevsky, and Vincent Vanhoucke. 2015. Pedestrian detection with a large-field-of-view deep network. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 704–711.

[4] Tianqi Chen et al. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

[5] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[6] Ronan Collobert et al. 2002. *Torch: a modular machine learning software library*. Technical Report. Idiap.

[7] Mehdi Goli. 2017. SYCL backend for Eigen. *Technical peresentation in 1st workshop on Distributed and Heterogeneous Programming in C and C++(DHPCC++17)- To be appear in May 2017* (2017).

[8] Mehdi Goli, Luke Iwanski, and Andrew Richards. 2017. Accelerated Machine Learning Using TensorFlow and SYCL on OpenCL Devices. In *Proceedings of the 5th International Workshop on OpenCL*. ACM, 8.

[9] OpenACC Working Group et al. 2011. The OpenACC Application Programming Interface. (2011).

[10] Junli Gu, Yibing Liu, Yuan Gao, and Maohua Zhu. 2016. OpenCL caffe: Accelerating and enabling a cross platform machine learning framework. In *Proceedings of the 4th International Workshop on OpenCL*. ACM, 8.

[11] Gael Guennebaud, Benoit Jacob, et al. 2014. Eigen: a C++ linear algebra library. (2014).

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[13] Yangqing Jia et al. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 675–678.

[14] Microsoft. 2013. C++ AMP: Language and Programming Model. (2013).

[15] CUDA Nvidia. 2010. Programming guide. (2010).

[16] ARB OpenMP. 2011. OpenMP Application Programming Interface. (2011).

[17] Ben Sander, Greg Stoner, Siu-chi Chan, WH Chung, and Robin Maffeo. 2015. HCC: A C++ Compiler For Heterogeneous Computing. *HSA Foundation, Tech. Rep.* (2015).

[18] Khronos OpenCL Working Group SYCL subgroup. 2015. SYCL Specification. (2015).