Efficient Sparse Deep Neural Network Computation on GPU

Lillian Wang
LillianWang@my.unt.edu
University of North Texas
Denton, TX, USA

Avik Malladi
AvikMalladi@my.unt.edu
University of North Texas
Denton, TX, USA

Yuede Ji
yuede.ji@unt.edu
University of North Texas
Denton, TX, USA

Abstract

This paper presents GPU optimizations using TVM in response to the MIT/IEEE/Amazon GraphChallenge.org Sparse Deep Neural Network (SpDNN) 2022 Challenge. Although various deep neural network models exist, SpDNNs have shown great improvements in the size and memory of neural networks. SpDNNs provide unique scalability difficulties in which optimizations and advancements can be made. Apache TVM is a machine learning compiler framework for CPUs and GPUs. It has been shown to have promising improvements for the performance, deployment, and optimizations of the networks. To evaluate its effectiveness for SpDNNs, this work builds SpDNNs with Apache TVM and compares with the GraphChallenge baseline code. When testing with the GraphChallenge dataset, TVM-based implementation can achieve promising optimization opportunities.

1 Introduction

Deep neural networks (DNN) are an increasingly valuable machine learning method that uses representation learning to generate predictions [1]. DNNs are able to generate representative features by using more hidden layers, which as a result makes DNNs more accurate. Similarly, given that certain inputs are *sparse*, and many of the weights are 0, storage and computation of inputs can be done far more efficiently both in space and time.

Here, sparse refers to the fact that only a few percent of connections exist between layers. In other words, most of the connection weights are 0. Therefore, improving the computation performance of SpDNNs from CPUs and GPUs can provide a significant insight on how neural networks can continue to get faster and more efficient with fewer parameters or data. Improving the computation efficiency of SpDNNs can produce a wide array of applications. Further applications in this can lead to various sparse software and hardware resources.

1.1 Key Insights and Contributions

We aim to improve the computation efficiency of SpDNN by using Apache TVM, a machine learning compiler that allows code to easily run in both CPUs and GPUs [3].

In summary, we make two contributions:

- Our TVM implementation of SpDNN inference allows our model to be easily deployed with both CPU and GPU. Further, we optimize the inference process by partitioning and parallelizing the model with TVM's scheduling services.
- The experiment results on a variety of testing data and layers show that our implementation is effective in optimizing SpDNNs. In particular, on the GPU, our implementation can achieve 1.6× speedup over PyTorch-based implementation. Also, on CPU, we can also achieve 1.6× speedup over the official GraphChallenge MATLAB code.

2 Background and Motivation

Problem Definition. The sparse deep neural network graph challenge involves timing the deep neural network with given MNIST input data [2] [4]. Additionally, this output is also to be compared to the provided truth categories. Provided from GraphChallenge.org, given the weight matrices W_l, sparse input data Y_0, we load our generated deep neural network and its inputs. We also include a bias vector B_l to match the input. The process of reading the weight matrices and identifying the number of edges or nonzero values is timed. Next, for each hidden layer in the deep neural network, we compute the next layer using the following equation:

$$Y_{l+1} = ReLU(Y_l \cdot W_l + B_l) \tag{1}$$

We time the computation of the equation as the inputs pass each layer of the deep neural network. Finally, we check for accuracy with given output data and find the rate for the neural network inference as follows:

$$\frac{(\text{number of inputs})(\text{number of connections})}{\text{time(sec)}.}$$

3 Overview

Figure 1 overviews the TVM implementation of the SpDNN inference. First, in order to construct a model for the DNN inference, we define the inference equation using Tensor Expression. Tensor Expression is a lower level expression namespace in TVM which allows us to create an intermediate representation (IR) template. Second, we use TVM's schedule primitives to optimize the IR template and parallelize the partitions. The result of this step is a tensor-level IR (TIR) template which TVM processes automatically into machine

PL'18, January 01–03, 2018, New York, NY, USA
2018.

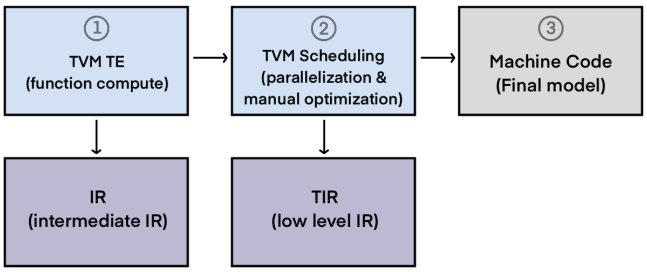

Figure 1. Overview of the TVM implementation of the SpDNN inference.

	Layers	Edges	Time (s)	Edges/second
TVM			7.24	5.431160e+05
MATLAB	120	3,932,160	0.82	47.933179e+05
Sparse library			7.29	5.393909e+05
TVM			29.39	5.351274e+05
MATLAB	480	15,728,640	3.25	48.468130e+05
Sparse library			28.78	5.465262e+05
TVM			115.42	5.450974e+05
MATLAB	1920	62,914,560	13.26	47.456308e+05
Sparse library			158.63	3.965995e+05

Table 1. Performance for reading sparse weight matrices with 1024 neurons.

	Layers	Edges	Time (s)	Edges/Second
TVM			46.19	5.107807e+09
MATLAB	120	3,932,160	72.42	3.257796e+09
Sparse library			43.09	5.475275e+09
TVM			184.07	5.126857e+09
MATLAB	480	15,728,640	289.61	3.258636e+09
Sparse library			163.13	5.785218e+09
TVM			733.46	5.146666e+09
MATLAB	1920	62,914,560	1220.09	3.093986e+09
Sparse library			658.55	5.732124e+09

Table 2. Single CPU performance for the deep neural network computation with 1024 neurons and varying layer size.

	Layers	Edges	Time (s)	Edges/Second
TVM	120	3,932,160	42.56	5.551285e+09
Sparse library			68.99	3.419765e+09
TVM	480	15,728,640	172.91	5.457899e+09
Sparse library			275.97	3.419645e+09
TVM	1920	62,914,560	717.60	5.260383e+09
Sparse library			1105.53	3.414547e+09

Table 3. Single GPU performance for the deep neural network computation with 1024 neurons and varying layer size.

code. Later, the completed model takes the weight matrix and input feature vectors as input and outputs the product with the defined operations and optimizations.

4 Experiment

The experiments are performed on a server with two Intel Xeon Silver 4309Y CPUs each containing 8 cores. The server runs Rocky Linux 8.6 with hyper-threading enabled. The server also has one A40 NVIDIA GPU with 48GB and is running CUDA Toolkit 11.7. We test our implementation with two baselines: the GraphChallenge MATLAB code, a basic SpMM and activation function (CPU), and our sparse library implementation (uses CuPy, a SciPy and NumPy to run in GPU).

Dataset. We use the datasets from the GraphChallenge [4]. The weight matrices for the DNNs contain 1,024, 4,096, 16,384, and 65,536 neurons, respectively, with up to 1,920 layers. Feature vectors are from MNIST and contain photos of handwritten numbers. A collection of 60,000 feature vectors are provided for each number of neurons.

Data reading performance. Table 1 describes the results for reading in the weight matrices. Both the TVM implementation and the sparse library implementation use the same function to read matrices, so the run times are similar. However, MATLAB appears to be more specialized for reading from files and is faster than other implementations by an average factor of 9.4×.

CPU performance. We first examine the performance of TVM for CPU as shown in Table 2. The TVM implementation performs faster than the MATLAB implementation with an average 1.6× speedup. The sparse library has a similar speedup of 1.8×. The cause of the speedups is likely due to the higher efficiency of the optimizes sparse matrix operations in the libraries, although the parallelization that TVM provides also increases the efficiency. Our implementation is slightly slower than the sparse library by a factor of 1.1× . This is most likely due to the sparse library's use of SciPy. Since the MNIST datasets are often sparse, using CSR matrices takes advantage of this sparsity and increases the efficiency. When the sparse library is modified so that the feature vectors are dense, the run time becomes slower than the TVM run time by roughly 6 fold. This means that even without accommodating the sparsity of the input vectors, our implementation with TVM is able to keep up with the sparse library.

GPU performance. When using GPU, the TVM implementation outperforms the sparse library by an average of 1.6×. This indicates that TVM is more effective in utilizing the GPU architecture than CuPy. TVM's block and thread scheduling may be more efficient than the optimizations that CuPy provides in the sparse library. It should also be noted that the GPU does not perform much better than the CPU. The TVM implementation only has a 1.1× speedup for GPU and the sparse library is slower than the CPU implementation by 1.7×.

5 Conclusion

The sparse Deep Neural Network Graph Challenge aims to improve upon the inference of sparse deep neural networks. We do this through Apache TVM, a machine learning compiler framework for various computer architectures. TVM's scheduling and tuning optimizations improve upon the GraphChallenge baseline and shows promise compared to other sparse librarys. Further research may be done to apply TVM's autoTVM or AutoScheduler to tune the inference, explore TVM's accommodations for other Python machine learning libraries such as Pytorch and TensorFlow, or learn how to optimize multiplication algorithm for multiplication between two sparse matrices.

References

[1] Klaus-Robert Müller Grégoire Montavon, Wojciech Samek. 2018. Methods for interpreting and understanding deep neural networks. In *Digital Signal Processing, Volume 73*. ScienceDirect, 1–15.

[2] Mert Hidayetoğlu, Carl Pearson, Vikram Sharma Mailthody, Eiman Ebrahimi, Jinjun Xiong, Rakesh Nagi, and Wen-mei Hwu. 2020. At-Scale Sparse Deep Neural Network Inference With Efficient GPU Implementation. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7.

[3] Yuwei Hu, Zihao Ye, Minjie Wang, Jiali Yu, Da Zheng, Mu Li, Zheng Zhang, Zhiru Zhang, and Yida Wang. 2020. Featgraph: A flexible and efficient backend for graph neural network systems. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–13.

[4] Jeremy Kepner, Simon Alford, Vijay Gadepally, Michael Jones, Lauren Milechin, Ryan Robinett, and Sid Samsi. 2019. Sparse deep neural network graph challenge. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7.