5-3-2022

Extending Tensor Virtual Machine to Support Deep-Learning Accelerators with Convolution Cores

Yanzhao Wang
Portland State University, wyanzhao@pdx.edu

Fei Xie
Portland State University, xie@pdx.edu

Citation Details

Published as: Wang, Y., & Xie, F. (2022, March). Extending Tensor Virtual Machine to Support Deep-Learning Accelerators with Convolution Cores. In 2022 26th International Conference on Engineering of Complex Computer Systems (ICECCS) (pp. 189-194). IEEE.

Extending Tensor Virtual Machine to Support Deep-Learning Accelerators with Convolution Cores

Yanzhao Wang and Fei Xie
Dept. of Computer Science, Portland State University, Portland, OR 97229, USA
{*wyanzhao, xie*}*@pdx.edu*

***Abstract*—Deep-learning accelerators are increasingly popular. There are two prevalent accelerator architectures: one based on general matrix multiplication units and the other on convolution cores. However, Tensor Virtual Machine (TVM), a widely used deep-learning compiler stack, does not support the latter. This paper proposes a general framework for extending TVM to support deep-learning accelerators with convolution cores. We have applied it to two well-known accelerators: Nvidia's NVDLA and Bitmain's BM1880 successfully. Deep-learning workloads can now be readily deployed to these accelerators through TVM and executed efficiently. This framework can extend TVM to other accelerators with minimum effort.**

***Index Terms*—Application Software, Software Architecture, Artificial Intelligence, Open Source Software**

I. Introduction

Traditional CPUs and GPUs are not meeting the growing computation and energy efficiency demands of various applications such as the Internet of Things (IoT). As a result, dedicated deep-learning accelerators have become popular in training and inference applications [1]. There are two major accelerator architectures [2]: one based on general matrix multiplication (GEMM) units and the other based on convolution cores. Google Edge TPU [3] and VTA [4] represent the GEMM-unit based architecture, often referred to as TPU-like accelerators. Nvidia's NVDLA [5] and Alibaba Hanguang 800 [6] represent the convolution-core-based architecture.

To deploy deep-learning workloads to accelerators, accelerator manufacturers provide users with proprietary compilers or software development kits (SDKs) that usually support a limited number of deep-learning model formats. For example, NVDLA's compiler only supports the Caffe [7] model format. Models in other popular neural network formats, such as TensorFlow [8] and MxNet [9], are not readily deployable to NVDLA. Previous studies have attempted to enable additional model formats on NVDLA. For example, ONNC [10] works on NVDLA and Bitmain Sophon accelerators [11], offering support for the ONNX portable model format [12]. However, it fails to offer generic model supports for various accelerators.

Tensor Virtual Machine (TVM) [13] is a well-known, generally applicable, and versatile end-to-end deep-learning compiler stack. It supports various front-end frameworks such as MxNet and TensorFlow. It also supports multiple hardware backends, including CPUs, GPUs, and TPU-like accelerators such as Google Edge TPU and VTA. However, TVM does not readily support accelerators with convolution cores. This paper identifies and addresses two challenges that caused the incompatibility between TVM and such accelerators:

1) *Intrinsic incompatibility between TVM low-level IR and instruction set architectures (ISAs) of convolution-core-based accelerators*: TVM low-level IR supports GEMM operations such as matrix multiplication, while the ISAs of convolution-core-based accelerators typically support higher-level graph operations such as convolution.
2) *Loss of hardware-related optimization opportunities*: TVM's high-level computation graph is hardware independent; directly mapping it to accelerator ISAs misses out on hardware-related optimization opportunities.

We propose a general framework for extending the TVM compiler stack to support accelerators with convolution cores and have made the following three contributions:

1) Identified challenges that hinder TVM's support of accelerators with convolution cores by analyzing abstraction mismatches between the IRs of TVM and the ISAs of these accelerators;
2) Developed three new extensions for TVM to support convolution-core-based accelerators: (1) a convolution-core-oriented intermediate representation (IR), (2) a supplemental operator dependency graph, and (3) an accelerator-specific support library;
3) Evaluated the applicability and efficiency of our TVM extensions by applying it to two accelerator backends, Nvidia's NVDLA and Bitmain's BM1880. Our extended TVM outperforms Nvidia's official NVDLA compiler and ONNC in both model execution time and memory consumption.

The remainder of this paper is organized as follows. Section 2 identifies the challenges preventing convolution-core-based accelerators from working with TVM. Section 3 presents our framework for extending TVM to such accelerators: NVDLA and BM1880. Section 4 discusses experimental results from extending TVM to two accelerators. Section 5 presents related works. Section 6 concludes and discusses future work.

II. Challenges

This section analyzes the challenges that intrinsically stem from the mismatches between TVM IRs and the ISAs of accelerators with convolution cores. TVM currently supports three kinds of hardware backends: CPUs, GPUs, and TPU-like accelerators [13], which have RISC-like ISAs. As shown

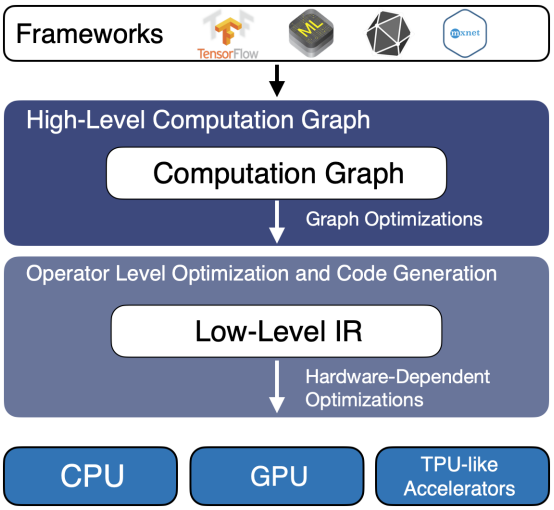

Figure 1. Overview of TVM. TVM supports multiple deep-learning framework frontends, e.g., MxNet and TensorFlow, and targets at multiple backends, e.g., CPU, GPU, and TPU-like accelerators.

Table I
DESCRIPTION OF TVM COMPUTATION GRAPH NODES.

Operators	Description
Conv	Y[b, c, w] = $\sum_{dw,k}$ data$[b, k, \text{strides}[0]*w+dw]*$ weight$[c, k, dw]$
Dense	Y = X * $W^T + bias$
Relu	Y = max(x, 0)
BiasAdd	Y = X + bias
Softmax	$Y(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)}$

in Figure 1, TVM implements optimizations on two levels for these backends: hardware-independent optimizations at the higher computation graph level and hardware-dependent ones at the lower IR level. Table I describes TVM computation graph nodes. The low-level TVM IR is similar to the RISC-like ISAs of TVM's original hardware backends in abstraction levels. We take the Dense node as an example. During the lowering process, TVM transforms the Dense node into RISC-like low-level IR nodes such as Add, Sum, Multiply and Cast, through the following code:

```
auto k = tvm::te::reduce_axis(Range(0, in_dim),"k");
auto matmul = tvm::te::compute({batch, out_dim},
    [&](Var i, Var j) {
        return tvm::sum(tvm::cast(out_dtype, data(i,
     k)) * tvm::cast(out_dtype, weight(j, k)), {k});
      }, "tensor", "dense");
if (bias.defined()) {
    matmul = tvm::te::compute({batch, out_dim},
        [&](Var i, Var j) { return matmul(i, j) +
    tvm::cast(out_dtype, bias(j)); }, "tensor",
    kBroadcast);
  }
```

Accelerators with convolution cores, such as NVDLA [5] and Alibaba Hanguang 800 [6], have specialized hardware units to process high-level deep-learning operators. Their ISAs are more CISC-like and have similar abstraction level as high-level neural network operators, such as Conv and Relu, as shown in Table II. Unfortunately, these supported operators

Table II
DESCRIPTION OF NVDLA HARDWARE UNITS. NVDLA HAS SPECIALIZED HARDWARE UNITS TO PROCESS HIGH-LEVEL DEEP-LEARNING OPERATORS.

NVDLA Units	Supported Operators
BDMA	Shuffle, Concat, Split
CONV	Conv, FullyConnected
SDP	Relu, BatchNorm
PDP	AvgPool, MaxPool
CDP	LRN

Table III
TVM PROVIDES GROUPS OF HARDWARE-DEPENDENT AND HARDWARE-INDEPENDENT GRAPH OPTIMIZATIONS.

Hardware-Independent Graph Optimizations	Hardware-Dependent Optimizations
Operator Fusion	Cache Location
	Parallel Cooperation
Data Layout Transformation	Thread Biding
	Latency Hiding

are at a higher abstraction level than TVM low-level IR, which blocks code generation during the lowering process.

Taking NVDLA as an example. As Figure 2 shows, since NVDLA implements many operations in hardware internals, such as matrix multiplication and addition within the convolution cores, NVDLA's ISA is at a relatively higher abstraction level. Its abstraction level largely resembles TVM's high-level computation graph, making it incompatible with TVM low-level IR. This hinders code generation from TVM lower-level IR to NVDLA's ISA. Therefore, the **first challenge** that we address is the intrinsic incompatibility between TVM low-level IR and the ISAs of convolution-core-based accelerators.

The **second challenge** is the loss of optimization opportunities. As Table III shows, TVM provides groups of hardware-dependent and hardware-independent optimizations strategies. Particularly, TVM low-level IR enables hardware-dependent optimization, and computation graph enables hardware-independent optimizations. However, if we merely address the first challenge, the TVM extensions still cannot achieve desired performance since TVM's original hardware-dependent optimization strategies do not apply to convolution-core-based accelerators. Besides, these accelerators have diverse micro-architectures, and no universal strategy can fully exploit optimization opportunities for all. Thus, it is necessary to design two new IRs: (1) an IR providing universal supports to all convolution-core-based accelerators with basic optimizations, and (2) a customized IR that provides different target accelerators with tailored optimization strategies to exploit specific optimization opportunities further.

III. OUR APPROACH

This section presents a general framework for extending TVM to support accelerators based on convolution cores with minimal modifications. We illustrate this framework with NVDLA, a popular and representative accelerator based on convolution cores. Figure 3 illustrates our extended TVM

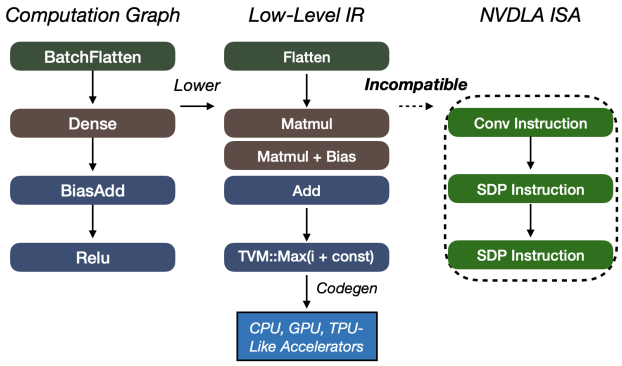

Figure 2. Illustration of the incompatibility issue. NVDLA has a high-level ISA since it implements many operations in hardware internals, such as matrix multiplication and addition within the convolution cores. It is incompatible with TVM's low-level IR.

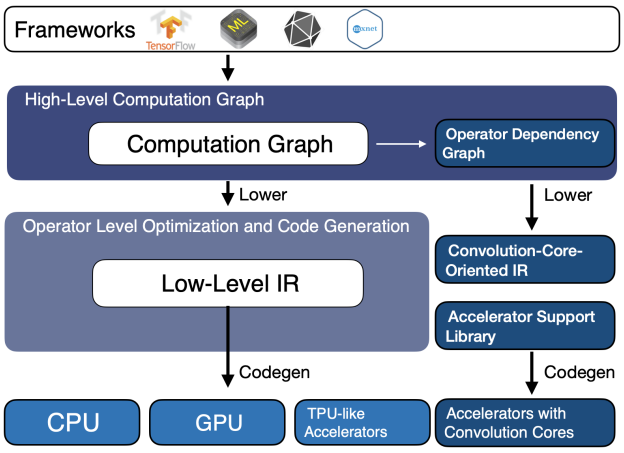

Figure 3. Overview of the extended TVM stack. We add three extensions: an operator dependency graph, a convolution-core-oriented IR, and an accelerator support library. With these extensions, TVM can work with accelerators based on convolution cores.

framework:

1) *A new convolution-core-oriented IR* in the operator-level optimization and code generation stage. It addresses the incompatibility between TVM and the ISAs of convolution-core-based accelerators. Optimizations generically applicable to all convolution-core-based accelerators are also conducted on this IR.
2) *An operator dependency graph* in the high-level computation graph stage. This graph preserves dependency information among neural network operators necessary for lowering the high-level computation graph to the convolution-core-oriented IR.
3) *An accelerator support library*. This library maps the convolution-core-oriented IR to the accelerator ISAs and completes code generation. Accelerator-specific low-level optimizations are also conducted in this stage.

Below we present the details of our TVM extensions.

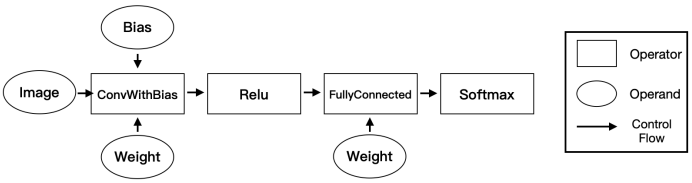

Figure 4. Example of the convolution-core-oriented IR. Since accelerators with convolution cores usually require dependency information between operators to control the execution order, our IR adopts a tree structure to maintain the dependency between nodes. The IR graph's nodes represent operators and the edges represent operands, such as weights and inputs.

A. Convolution-Core-Oriented IR and Optimizations

There are two reasons why we need the convolution-core-oriented IR. First, it eliminates the incompatibility between TVM IR and the ISAs of convolution-core-based accelerators. When target devices are convolution-core-based accelerators, such as NVDLA, TVM lowers the computation graph into the new IR instead of the original low-level GEMM-oriented IR. Second, this new IR provides an adequate abstraction for optimizations commonly applicable to all convolution-core-based accelerators.

We design this convolution-core-oriented IR at the higher neural network operator level. It includes IR nodes such as ConvWithBias, Relu, and BatchNorm. Figure 4 shows an example of the IR consists of a one-layer convolution neural network. Since accelerators with convolution cores usually require dependency information between operators to control the execution order, our IR adopts a tree structure to maintain the dependency between nodes. In this IR, nodes represent operators, and edges represent operands, such as weights, inputs, and biases.

Based on this IR, we implement hardware-independent optimization strategies applicable to all convolution-core-based accelerators. The optimizations that are currently conducted on this IR include:

1) Elimination of operators unused in inference workloads, such as *Dropout*;
2) Pre-computation of operators at compile-time, such as *Reshape*;
3) Division of large *GlobalAvgPool* into multiple *AvgPool* whose kernel sizes are within the hardware resource limits of convolution-core-based accelerators.

Comparing our convolution-core-oriented IR with TVM high-level computation graph, there are two major differences:

1) TVM's computation graphs are hardware-independent, incompatible with hardware-dependent optimizations. Our IR supports optimizations generically applicable to all accelerators with convolution-cores.
2) The design of our convolution-core-oriented IR closely matches accelerator hardware, which will facilitate code generation for accelerators with convolution cores.

TVM provides developers a convenient way to add new hardware backends to reduce engineering efforts. We utilized this feature and registered our IR to TVM as an external

backend to minimize our modifications to TVM. When target devices are convolution-core-based accelerators, e.g., NVDLA, TVM will invoke this new backend to lower the computation graph to the new convolution-core-oriented IR.

B. Operator Dependency Graph

Originally, TVM does not explicitly pass down operator dependency information, particularly control dependencies, when it lowers a model from the high-level computation graph to the lower-level GEMM-oriented IR. The GEMM-oriented IR will only capture data dependency information, missing control dependencies. However, the convolution-core-oriented IR needs complete dependency information to operate, and reconstructing such dependencies at the lower level would be challenging and inefficient. Therefore, we add an extra compiler pass in TVM's high-level computation graph to preserve and pass on all dependency information when TVM lowers the model to the convolution-core-oriented IR.

C. Accelerator Support Library

We complete code generation and hardware-dependent optimizations in the accelerator support library. We use NVDLA to demonstrate the workflows. Since the convolution-core-oriented IR is still coarse-grained, we introduce an additional hardware-dependent IR to maximize optimizations in the support library. As Figure 5 shows, we design the library's IR nodes, such as CONV, SDP, and PDP, on the hardware-unit level to match NVDLA's units. Multiple deep-learning operators such as BatchNorm, Relu, and Multiply will utilize the SDP unit. Since the SDP unit supports multiple data access ports, if adjacent operators utilize the SDP unit and their data ports are not in conflict, they can be fused into one SDP operator. Another example is that for two adjacent ALU operators, their computation may be fused into a new ALU operator, such as fusing two Adds into one. In both cases, we can reduce model execution time and memory accesses. The NVDLA-specific optimizations currently supported include:

1) Fusion of ALU operators at compile-time;
2) Fusion of adjacent SDP operators;
3) Elimination of extra CONV core memory accesses.

As Figure 5 shows, with our extensions, TVM can work with accelerators with convolution cores. Next section demonstrates how to deploy neural networks on such accelerators through TVM, using NVDLA and BM1880 as examples.

IV. Evaluations

This section evaluates our work from five aspects: supported TVM operators, supported deep-learning models, performance results, memory usage results, and lines of code (LoC) for adapting TVM to a new convolution-core-based accelerator. We perform all experiments on a desktop with a 12-core AMD Ryzen 5900x CPU, 128 GB of RAM, and Ubuntu 18.04 OS. We target Nvidia's NVDLA and Bitmain's BM1880.

Table IV
Comparison with the official NVDLA compiler.

Operators	Official NVDLA Compiler	Our TVM NVDLA Extensions	Our TVM BM1880 Extensions
ConvWithBias	✓	✓	✓
Relu	✓	✓	✓
LRN	✓	✓	✓
Dropout	✓	✓	✓
Transpose	✓	✓	✓
BatchFlatten	✓	✓	✓
FullyConnected	✓	✓	✓
BiasAdd	✓	✓	✓
Softmax	✓	✓	✓
Multiply	✓	✓	✓
BatchNorm	✓	✓	✓
GlobalAvgPool	✓	✓	✓
AvgPool	✓	✓	✓
MaxPool	✓	✓	✓
Concatenate	✓	✓	✓
Reshape	✗	✓	✓
Add	✗	✓	✓
Copy	✗	✓	✓

Table V
Supported deep-learning models.

Deep-learning Models	Official NVDLA compiler	Our TVM NVDLA Extensions	Our TVM BM1880 Extensions
AlexNet	✓	✓	✓
CaffeNet	✓	✓	✓
ResNet18	✓	✓	✓
GoogLeNet	✗	✓	✓

A. Supported Deep-Learning Operators

Table IV shows all operators we currently support. We compared our work with the official NVDLA compiler. The first column lists operator names. The second column indicates the operators supported by the official NVDLA compiler. The third and fourth columns show the operator support status on NVDLA and BM1880 using our extended TVM stack. Our extensions support more deep-learning operators, e.g., Reshape and Add, than the official NVDLA compiler. Some operators can be pre-computed during compile-time, e.g., Reshape, instead of running on the hardware side. The official NVDLA compiler does not support such optimizations.

B. Supported Deep-Learning Models

As deep convolution neural networks (CNNs) are prevalent in deep-learning workloads , we select classic CNNs to assess performance. As Table V shows, we can deploy AlexNet [14], CaffeNet [7], GoogLeNet [15] and ResNet18 [16] to NVDLA and BM1880 through our extended TVM stack, while the official NVDLA compiler does not support GoogLeNet.

C. Preliminary Performance Evaluations

We compared our framework's performance results with the official NVDLA compiler and ONNC. They are denoted as NVDLA on TVM, the NVDLA official compiler, and ONNC. For each inference workload, we conducted experiments five

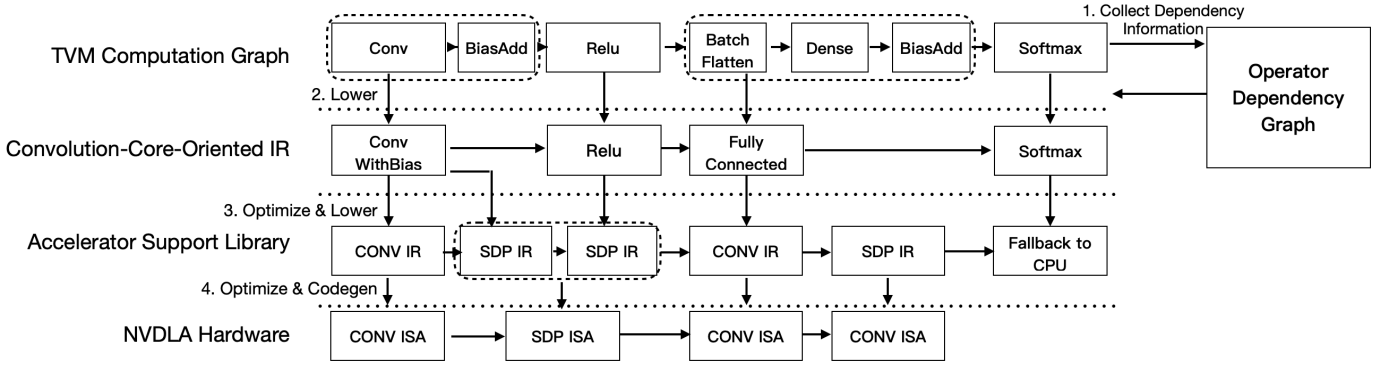

Figure 5. Overview of the end-to-end workflow for our extensions on NVDLA. First, we collect the operator dependency graph information. With this information, we can lower TVM graph IR to the convolution-core-oriented IR. Some TVM graph IR nodes, such as Conv and its following BiasAdd, can be fused into our new ConvWithBias Node. At this stage, we can conduct optimizations that are shared among all accelerators with convolution cores. Then we lower the convolution-core-oriented IR to the hardware-specific IR in the accelerator support library. We can conduct hardware-related optimizations at this stage, such as fusing two SDP operators into one SDP instruction to improve performance, and complete code generation.

times and calculated the averages. For the NVDLA official compiler, since it only supports Caffe models, we used the models from the Caffe model zoo. For ONNC and our framework on TVM, since they both support the ONNX format, we used the same models from the ONNX model zoo.

For quick evaluation purposes, currently we utilize the NVDLA virtual platform (VP) [17], a SystemC [18] simulator, as the hardware target. For the BM1880 platform, which uses a text file to save instructions, we compared the generated text file with ONNC for correctness verification. We will evaluate BM1880's performance on actual Bitmain hardware in the future. As Figure 6 illustrates, our framework achieves the best results in each deep-learning inference workload, because our work supports both coarse-grained and fine-grained optimizations. In comparison, ONNC can only conduct coarse-grained optimizations. The official NVDLA compiler, in most cases, achieves better performance than ONNC, since it supports fine-grained optimizations. Since the loadable file generated by our work can run on the official NVDLA VP, our work is readily portable to NVDLA hardware implementations, such as FPGA, for further evaluation as part of future work.

D. Preliminary Memory Usage Evaluations

We compared the memory usage in the loadable file generated by our framework, the official NVDLA compiler, and ONNC. As Figure 7 shows, in the AlexNet and CaffeNet tests, our work uses the least amount of memory in the loadable file. For GoogLeNet, our framework uses less memory than ONNC. In the ResNet18 test, the official NVDLA compiler uses the least amount of memory.

E. Scalability Evaluations

Since our goal is to make TVM support various convolution-core-based accelerators, it is important to evaluate the development efforts required to adapt our framework to a new accelerator. Our extensions require minimal modifications to TVM. Since we made no changes to the existing TVM files, it is easy to patch our work to TVM's mainline codes. The

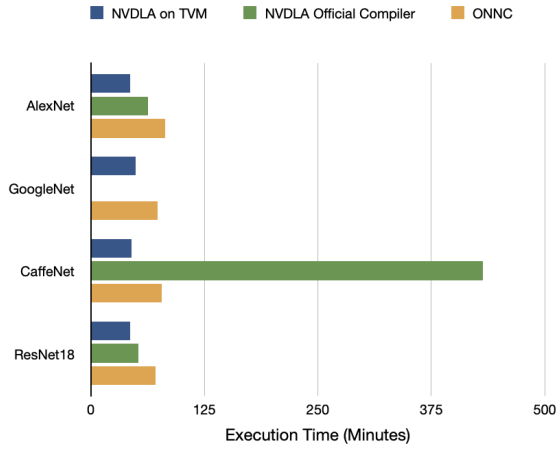

Figure 6. Performance comparison between our TVM extensions for NVDLA, the NVDLA official compiler and ONNC. Our work performs the best. Since the official NVDLA compiler does not support GoogLeNet, it has no data on that row.

extensions have three parts. The first part is IR registration, adding the extensions as a new pattern to TVM, so TVM can correctly lower its high-level computation graph to our convolution-core-oriented IR. The LoC for this part is *75*. The second part is the convolution-core-oriented IR and its optimizations, and the LoC for this part is *4943*. The third part is the hardware-specific accelerator support library, the only non-reusable component that needs to be built specifically for every new target accelerator. We use man-week as the unit to evaluate efforts spent on building the support library. It took *3 man-weeks* to complete our modification specific for NVDLA. For BM1880, it took *2 man-weeks*. The LoC for NVDLA is *2378*, and the LoC for BM1880 is *1663*.

V. Related Work

Comparing with the official NVDLA compiler [5], our work can support more deep-learning model formats through a powerful compiler stack, TVM. We implemented a convolution-core-oriented IR based on ONNC's IR [10], allowing TVM

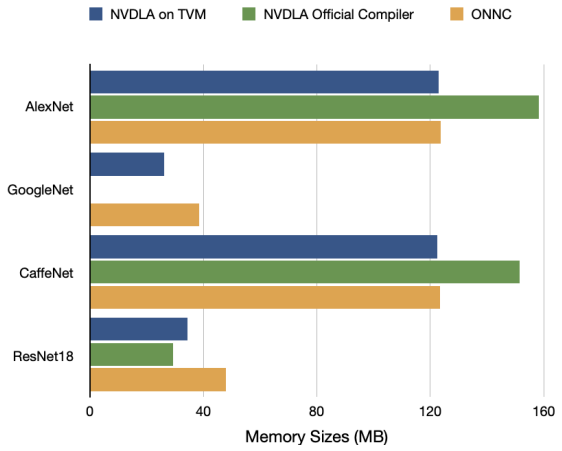

Figure 7. Memory usage comparison for the NVDLA loadable file between our TVM extensions for NVDLA, the NVDLA official compiler and ONNC. Our work uses the least amount of memory in AlexNet and CaffeNet. Since the official NVDLA compiler does not support GoogLeNet, it has no data on that row.

to support more operators and conduct high-level accelerator-independent optimizations efficiently. Furthermore, powered by optimizations on the hardware-specific IR within our accelerator support library, which are not available in ONNC, our TVM extensions significantly outperform ONNC.

There is another work attempting to integrate TVM and NVDLA [19]. In comparison, our work has four major advantages. (1) Our work implements multiple-level IR optimizations, while their work only supports TVM high-level optimizations. (2) Our work can directly use TVM to generate the NVDLA loadable file, while their work does not implement the end-to-end flow. They use TVM to dump a JSON file, and then use the NVDLA compiler to load this JSON file to finish code generation. (3) Our work supports multiple practical models, such as GoogLeNet, ResNet, CaffeNet, and AlexNet, while their model only supports LeNet [20]. (4) Our work has developed generic TVM extensions for all convolution-core-based accelerators (e.g., NVDLA and BM1880), while their work is only applicable to NVDLA.

VI. Conclusions and Future Work

This paper proposes a general framework for extending TVM to support convolution-core-based deep-learning accelerators. To bridge the gap between the TVM IR and the ISAs of convolution-core-based accelerators, we have developed three extensions to TVM, including a convolution-core-oriented IR, a high-level operator dependency graph, and an accelerator support library. We have realized this framework on Nvidia's NVDLA and Bitmain's BM1880 and successfully executed popular deep-learning models on them efficiently.

We will pursue the following two improvements in our future work. To increase the integration level of TVM and convolution-core-based accelerators, we plan to develop a new unified accelerator runtime system utilizing TVM's just-in-time compilation approach. Nvidia's original runtime system requires a deep-learning model to be converted to a binary format, NVDLA loadable, and then deploy the loadable file to hardware through its driver. This breaks a seamless integration flow and hinders data pre-processing on the TVM side. Another direction is to develop more optimization strategies that can leverage our convolution-core-oriented IR.

VII. Acknowledgment

This research is partially supported by Semiconductor Research Corporation Contract : 2932.001.

References

[1] W. G. Hatcher and W. Yu, "A survey of deep learning: Platforms, applications and emerging research trends," *IEEE Access*, vol. 6, pp. 24 411–24 432, 2018.

[2] T. L. Group, *The next generation of ai processors*. [Online]. Available: https://www.youtube.com/watch?v=Ke7hv3FPzi4, (accessed: 02-2021).

[3] *Edge tpu*. [Online]. Available: https://cloud.google.com/edge-tpu, (accessed: 02-2021).

[4] T. Moreau, T. Chen, Z. Jiang, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Vta: An open hardware-software stack for deep learning," *arXiv preprint arXiv:1807.04188*, 2018.

[5] *Nvdla*. [Online]. Available: http://nvdla.org, (accessed: 02-2021).

[6] *Alibaba hanguang 800*. [Online]. Available: https://www.alibabacloud.com/blog/announcing-hanguang-800-alibabas-first-ai-inference-chip_595482, (accessed: 02-2021).

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[9] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[10] W.-F. Lin, D.-Y. Tsai, L. Tang, C.-T. Hsieh, C.-Y. Chou, P.-H. Chang, and L. Hsu, "Onnc: A compilation framework connecting onnx to proprietary deep learning accelerators," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, IEEE, 2019, pp. 214–218.

[11] *Bitmain-sophon*. [Online]. Available: https://www.sophon.ai/, (accessed: 02-2021).

[12] *Onnx*. [Online]. Available: https://onnx.ai/, (accessed: 02-2021).

[13] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, *et al.*, "{tvm}: An automated end-to-end optimizing compiler for deep learning," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 578–594.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[17] *Nvdla virtual platform*. [Online]. Available: http://nvdla.org/vp.html, (accessed: 02-2021).

[18] D. C. Black, J. Donovan, B. Bunton, and A. Keist, *SystemC: From The Ground Up*. Springer Science & Business Media, 2009, vol. 71.

[19] *Integrate-nvdla-and-tvm*. [Online]. Available: https://github.com/WuDan0399/Integrate-NVDLA-and-TVM, (accessed: 07-2021).

[20] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, 1989.