TVM: End-to-End Compilation Stack for Deep Learning

Tianqi Chen[1], Thierry Moreau[1], Ziheng Jiang[2,3], Haichen Shen[1]
Eddie Yan[1], Leyuan Wang[2,4], Yuwei Hu[5],
Luis Ceze[1], Carlos Guestrin[1], Arvind Krishnamurthy[1]
[1]Paul G. Allen School of Computer Science & Engineering, University of Washington
[2]Amazon Web Service, [3]Fudan University, [4]UC Davis, [5]TuSimple

1 INTRODUCTION

Deep learning models can now recognize images, process natural language, and defeat humans in challenging strategy games. The steadily advancing compute capabilities of modern hardware has played a prominent role in deep learning's present ubiquity and relevance in many problem domains. Many of the most popular deep learning frameworks, such as TensorFlow, MXNet, Caffe, and PyTorch, harness the power of modern hardware by focusing support on a narrow class of server-class GPU devices—with this support depending on the use of highly engineered and vendor-specific GPU libraries. However, the number and diversity of specialized deep learning accelerators is increasing rapidly in the wild. These accelerators pose an adoption challenge as they introduce new abstractions that modern compilers and frameworks are ill-equipped to deal with.

Providing support in various deep learning frameworks for diverse hardware back-ends in the present ad-hoc fashion is unsustainable. Ultimately, the goal is to easily deploy deep learning workloads to all kinds of hardware targets, including embedded devices, GPUs, FPGAs, and ASICs (e.g, the TPU), which significantly diverge in terms of memory organization, compute primitives etc. Given these requirements, the development of an optimization framework that can lower a high-level specification of a deep learning program down to low-level optimized code for any hardware back-end is critical.

Current deep learning frameworks rely on a computational graph intermediate representation to implement optimizations such as auto differentiation and dynamic memory management [3, 4, 7]. Graph-level optimizations, however, are often too high-level to handle hardware back-end-specific operator-level transformations. On the other hand, current operator-level libraries that deep learning frameworks rely on are too rigid and specialized to be easily ported across hardware devices. To address these weaknesses, we present TVM[1], (shown in Figure 1) an end-to-end system allowing the effective deployment of deep learning workloads specified in a high-level framework (including Caffe, MXNet, PyTorch, Caffe2, CNTK) to diverse hardware back-ends (including CPUs, GPUs, and FPGA-based accelerators).

2 FUNDAMENTAL CHALLENGES

An optimizing compiler for deep learning systems needs to expose both high-level and low-level optimizations. We summarize four fundamental challenges that TVM solves in this section:

High-level dataflow rewriting TVM exploits a computational graph representation to apply high-level optimizations. Computational

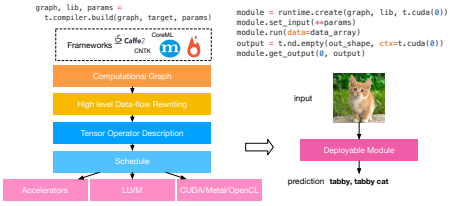

Figure 1: The TVM Stack Diagram. Current stack support: descriptions from many deep learning frameworks and targeting major CPU, GPU and specialized accelerators.

graphs provide a global view on computation tasks, yet avoid specifying how each computation task needs to be implemented. We can perform various high-level data-flow rewriting operations to optimize the computation. *Static memory planning* can be performed on the graph to pre-allocate memory to hold each intermediate tensor result. *Operator fusion* fuses multiple operators together into a single kernel without storing intermediate results back into global memory. We can also perform *data layout transformations* to use a more friendly data layout to speedup computation by exploiting a given hardware architecture's data layout constraints.

Memory reuse across compute units Modern GPUs and specialized accelerators have a shared memory organization for which the often used shared-nothing nested parallel model is not optimal. We enhanced the nested parallel model used in Halide [16] among other DSLs to allow thread cooperation via shared memory during computation. This enhancement, along with the nested parallel model, can generate GPU code that is performance-competitive with hand written kernels.

Tensorized compute intrinsics Deep learning workloads can be typically decomposed into tensor operators like matrix-matrix multiplication or 1-D convolution. These natural decompositions have led to novel hardware architectures that expose tensor compute primitives that go beyond vector-vector instructions. Examples include matrix-matrix multiplication [13], matrix-vector product [1] and 1D convolution [9]. These new primitives create novel challenges when scheduling high-level tensor operators: the schedule must leverage these primitives to benefit from hardware specialization. We dub this the *tensorization* problem, analogous to the vectorization problem in SIMD architectures.

Tensorization differs significantly from vectorization. The inputs to the tensor compute primitives are multi-dimensional, with fixed or variable lengths, and dictate different data layouts. More importantly, we cannot resort to a fixed set of primitives, as new deep learning accelerators are emerging with their own flavors of tensor instructions. Therefore, we need a solution that is *future proof* to support new generations of specialized architectures.

[1]An extended version of this paper can be found at https://www.cs.washington.edu/tr/2017/12/UW-CSE-17-12-01.pdf

SysML'18, Feb 2018,
2018.

Workload	MXNet(ms)	TVM (ms)	**Speedup**
ResNet18	1390	567	2.4
MobileNet	2862	209	12

Table 1: End-to-end experiment results on Raspberry Pi. TVM generates operators that outperform MXNet which is backed by hand-optimized libraries. The speedup on MobileNet demonstrates TVM's ability to quickly optimize emerging tensor operators, such as depth-wise [12] conv which are not supported in existing DNN libraries.

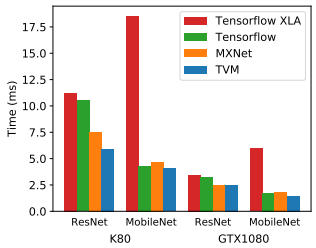

Figure 2: GPU end-to-end comparison of ResNet and MobileNet workloads among TVM, MXNet, Tensorflow, and Tensorflow XLA on NVIDIA Tesla K80 and GTX 1080.

To solve this challenge, we *separate the hardware interface from the schedule*. Specifically, we introduce a tensor intrinsic declaration mechanism. We use a tensor expression language to declare the behavior of each new hardware intrinsic, as well as the lowering rule associated with it. As a result, our tensorization procedure replaces a unit of computation with the corresponding tensor intrinsics to take advantage of hardware specialization.

Latency Hiding While traditional architectures with simultaneous multithreading and automatically managed caches implicitly hide latency in modern CPUs/GPUs, specialized accelerator designs usually favor leaner control and offload most of the scheduling complexity to the compiler stack. TVM can generate hardware explicit synchronization instructions to correctly interleave computation with memory operations. As a result, TVM can effectively hide the memory access latency and maximize the utilization and performance of the targeted accelerator.

Relation to Existing Works Deep learning frameworks [3, 4, 6, 7] provide convenient interfaces for users to run deep learning workloads. While existing frameworks currently depend on vendor specific libraries. they can leverage TVM's stack to generate optimized code a larger hardware devices. High-level computation graph DSLs are a typical way to represent and perform high-level optimizations. Tensorflow's XLA [3] and the recently introduced DLVM [19] fall into this category. While graph level representations are a good fit for high-level optimizations, they are too high-level to optimize tensor operators under a diverse set of hardware back-ends. Prior work that resorts to vendor crafted libraries require significant engineering effort for each hardware back-end and operator-variant combination.

Halide [16] introduced the principle of separation between compute and scheduling. We adopt Halide's insight and reuse its existing useful scheduling primitives in our compiler. The tensor operator scheduling is also related other works on DSL for GPUs [11, 17] as well as works on polyhedral-based loop transformation [5, 18].

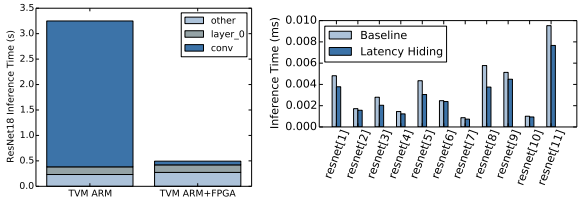

Figure 3: *Left*: We offload convolutions in the ResNet workload to VDLA, an FPGA-based accelerator design. The grayed-out bars correspond to layers that cannot be accelerated by the FPGA and therefore have to run on the CPU. The FPGA can provide a 40x acceleration on offloaded convolution layers over the ARM Cortex A9. The overall performance of the FPGA system is now bottlenecked by the CPU. We envision that extending the VDLA design to incorporate support for these other operators will help reduce inference time even further.
***Right*: The effect of latency hiding on a FPGA-based hardware accelerator design on each layer of ResNet**

TACO [14] introduces a generic way to generate sparse tensor operators on CPU. Weld [15] is a DSL for data processing tasks. We specifically focus on solving the new optimization challenges of optimizing deep learning workloads for GPUs and specialized accelerators. More importantly, we provide an end-to-end stack that can directly take descriptions from deep learning frameworks, and jointly optimize together with the high-level stack.

Despite the trend domain specific accelerators for deep learning [8, 13], it is yet unclear how a compilation stack can be built to effectively target these devices. TVM provides a generic solution to effectively target the specialized accelerators via tensorization and compiler-driven latency hiding.

3 EXAMPLE RESULTS

We evaluated TVM on three types of platforms—an embedded CPU, a server-class GPU, and a deep learning accelerator implemented on a low-power FPGA-based SoC. The benchmarks are based on real world deep learning inference workloads including ResNet [10] and MobileNet [12]. We compare our approach with existing deep learning frameworks including MxNet [7] and TensorFlow [2] that rely on highly engineered vendor-specific libraries. The results are summarized in Table 1, Figure 2 and Figure 3. TVM delivers performance across hardware back-ends that is competitive with state-of-the-art libraries for low-power CPU and server-class GPU.

The FPGA experiment demonstrates TVM's ability to target new hardware accelerator back-ends. It also shows the importance of latency hiding. Overall latency hiding achieves anywhere from 7% up to 54% latency reduction on each kernel by hiding some of the latency of loading data into the accelerator. In terms of compute resources, no latency hiding leads to at best 52% utilization, whereas latency hiding increases utilization to 74%.

4 CONCLUSION

Our system provides an end-to-end stack to solve fundamental optimization challenges across a diverse set of hardware back-ends. We hope our work can facilitate more studies of programming languages, system, and open new opportunities for hardware co-design techniques for deep learning systems.

REFERENCES

[1] 2017. NVIDIA Tesla V100 GPU Architecture: The Worldś Most Advanced Data Center GPU. (2017). http://www.nvidia.com/object/volta-architecture-whitepaper.html

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

[3] Martin 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)*. 265–283. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[4] Amit Agarwal, Eldar Akchurin, Chris Basoglu, Guoguo Chen, Scott Cyphers, Jasha Droppo, Adam Eversole, Brian Guenter, Mark Hillebrand, Ryan Hoens, Xuedong Huang, Zhiheng Huang, Vladimir Ivanov, Alexey Kamenev, Philipp Kranen, Oleksii Kuchaiev, Wolfgang Manousek, Avner May, Bhaskar Mitra, Olivier Nano, Gaizka Navarro, Alexey Orlov, Marko Padmilac, Hari Parthasarathi, Baolin Peng, Alexey Reznichenko, Frank Seide, Michael L. Seltzer, Malcolm Slaney, Andreas Stolcke, Yongqiang Wang, Huaming Wang, Kaisheng Yao, Dong Yu, Yu Zhang, and Geoffrey Zweig. 2014. *An Introduction to Computational Networks and the Computational Network Toolkit*. Technical Report MSR-TR-2014-112.

[5] Riyadh Baghdadi, Ulysse Beaugnon, Albert Cohen, Tobias Grosser, Michael Kruse, Chandan Reddy, Sven Verdoolaege, Adam Betts, Alastair F. Donaldson, Jeroen Ketema, Javed Absar, Sven van Haastregt, Alexey Kravets, Anton Lokhmotov, Robert David, and Elnar Hajiyev. 2015. PENCIL: A Platform-Neutral Compute Intermediate Language for Accelerator Programming. In *Proceedings of the 2015 International Conference on Parallel Architecture and Compilation (PACT) (PACT '15)*. IEEE Computer Society, Washington, DC, USA, 138–149.

[6] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. 2012. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop. (2012).

[7] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, , and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. In *Neural Information Processing Systems, Workshop on Machine Learning Systems (LearningSys'15)*.

[8] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*. IEEE Computer Society, Washington, DC, USA, 609–622. https://doi.org/10.1109/MICRO.2014.58

[9] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-efficient Dataflow for Convolutional Neural Networks. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 367–379. https://doi.org/10.1109/ISCA.2016.40

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity Mappings in Deep Residual Networks. *arXiv preprint arXiv:1603.05027* (2016).

[11] Troels Henriksen, Niels G. W. Serup, Martin Elsman, Fritz Henglein, and Cosmin E. Oancea. 2017. Futhark: Purely Functional GPU-programming with Nested Parallelism and In-place Array Updates. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017)*. ACM, New York, NY, USA, 556–571.

[12] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017). arXiv:1704.04861 http://arxiv.org/abs/1704.04861

[13] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[14] Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato, and Saman Amarasinghe. 2017. The Tensor Algebra Compiler. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 77 (Oct. 2017), 29 pages. https://doi.org/10.1145/3133901

[15] Shoumik Palkar, James J. Thomas, Deepak Narayanan, Anil Shanbhag, Rahul Palamuttam, Holger Pirk, Malte Schwarzkopf, Saman P. Amarasinghe, Samuel Madden, and Matei Zaharia. 2017. Weld: Rethinking the Interface Between Data-Intensive Applications. *CoRR* abs/1709.06416 (2017). arXiv:1709.06416 http://arxiv.org/abs/1709.06416

[16] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13)*. ACM, New York, NY, USA, 519–530.

[17] Michel Steuwer, Toomas Remmelg, and Christophe Dubach. 2017. Lift: A Functional Data-parallel IR for High-performance GPU Code Generation. In *Proceedings of the 2017 International Symposium on Code Generation and Optimization (CGO '17)*. IEEE Press, Piscataway, NJ, USA, 74–85.

[18] Sven Verdoolaege, Juan Carlos Juega, Albert Cohen, José Ignacio Gómez, Christian Tenllado, and Francky Catthoor. 2013. Polyhedral Parallel Code Generation for CUDA. *ACM Trans. Archit. Code Optim.* 9, 4, Article 54 (Jan. 2013), 23 pages.

[19] Richard Wei, Vikram Adve, and Lane Schwartz. 2017. DLVM: A modern compiler infrastructure for deep learning systems. *CoRR* abs/1711.03016 (2017). arXiv:1711.03016 https://arxiv.org/abs/1711.03016