From DSLs to Accelerator-Rich Platform Implementations: Addressing the Mapping Gap

Bo-Yuan Huang*† Steven Lyubomirsky*‡ Thierry Tambe*§ Yi Li†

Mike He‡ Gus Smith‡ Gu-Yeon Wei§ Aarti Gupta†

Sharad Malik† Zachary Tatlock‡

1 INTRODUCTION

In deep learning (DL), the hardware parallelism of accelerators like the GPU and the TPU has been used to efficiently implement tensor kernels, and further specialized devices continue to be developed to support this domain [6, 7]. However, while DL frameworks like TensorFlow [1] and TVM [2] have built-in support for specific accelerators like the TPU, adding support for a new custom device requires bespoke compiler extensions, which demand great effort and expertise in both the device and the compilation stack. For example, though the recent Bring Your Own Codegen (BYOC) [3] interface in TVM provides a means for invoking a custom accelerator without modifying TVM's internals, developers must implement a code generator that, given a portion of a TVM model that matches a syntactic pattern, produces C++ code that invokes the accelerator. Each BYOC integration is effectively a standalone compiler, requiring familiarity with both the target device and TVM.

Several factors complicate compiler support for accelerators:

- **Lack of a unified hardware specification**: Unlike the mostly stable, standardized specifications for general-purpose processors (ISAs), accelerators often have custom software-hardware interfaces via memory-mapped input/output (MMIO). The lack of a hardware semantics also makes it difficult to verify the compiler except by ad hoc testing.
- **Granularity mismatch between compiler intrinsics and accelerator operations**: For maximum power-performance efficiency, accelerators tend to implement coarse-grained operations (e.g., groups of DL model layers) in hardware, which may be difficult to map onto the source language's semantics.
- **Custom memory management and numerical representations**: Accelerators often utilize custom memory management schemes and new numerical representations for efficiency, which compilers must support and reason about when generating code.

We propose to address these challenges by using the Instruction-Level Abstraction (ILA) [5] to model both the semantics of hardware accelerators and compiler IR intrinsics. In particular, our goal is to enable the classic approach to building portable compilers, but extended to verifiable lowering to custom accelerators.

On the hardware side, an ILA model provides a *lifting* of the operations that an accelerator provides in the form of "instructions" that update the software-visible architectural state. This ISA-like interface for accelerators, enables software-hardware co-verification (with formal semantics) and provides a uniform interface for compilers that allows for drawing upon traditional compiling techniques for instruction selection. The formal ILA semantics for both the compiler intrinsics and accelerator operations enable formal verification of the mappings between them.

2 3LA: TVM + ILA + FLEXNLP

The proposed 3LA flow, shown in Figure 1, provides for application mapping from the TVM Relay IR to heterogeneous hardware including accelerators, such as the FlexNLP custom accelerator [9], using the ILA interface. 3LA is a model flow designed to be generalized across a broader range of domains.

2.1 TVM Relay

Relay is the top-level IR for TVM and provides a high-level interface for expressing DL models, including a type system for tensor shape information [8]. 3LA uses Relay to facilitate coarse-grained pattern matching and end-to-end reasoning about applications. In particular, Relay's tensor shape types help support generic code generation and facilitate memory planning. By working down from the Relay IR, our 3LA prototype supports any DL model format TVM can import, including MxNet, TensorFlow, ONNX, Keras, etc. In general, the 3LA methodology is fairly agnostic to the details of the source IR; it should be applicable to any source IR supporting syntactic pattern matching over application-level operators that can be aligned with target accelerator invocations.

2.2 ILA

The Instruction-Level Abstraction (ILA) is an ISA-like formal model for the functional behavior of accelerators at the architecture level or compiler IR intrinsics. ILA models have been developed and verified for several accelerators and general-purpose processors in the open-source ILAng platform [4] that supports modeling (in C++

*Equal contribution
†Princeton University, Princeton, New Jersey, USA
‡University of Washington, Seattle, Washington, USA
§Harvard University, Cambridge, Massachusetts, USA

This work was supported in part by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA.

LATTE '21, April 15, 2021, Virtual Event

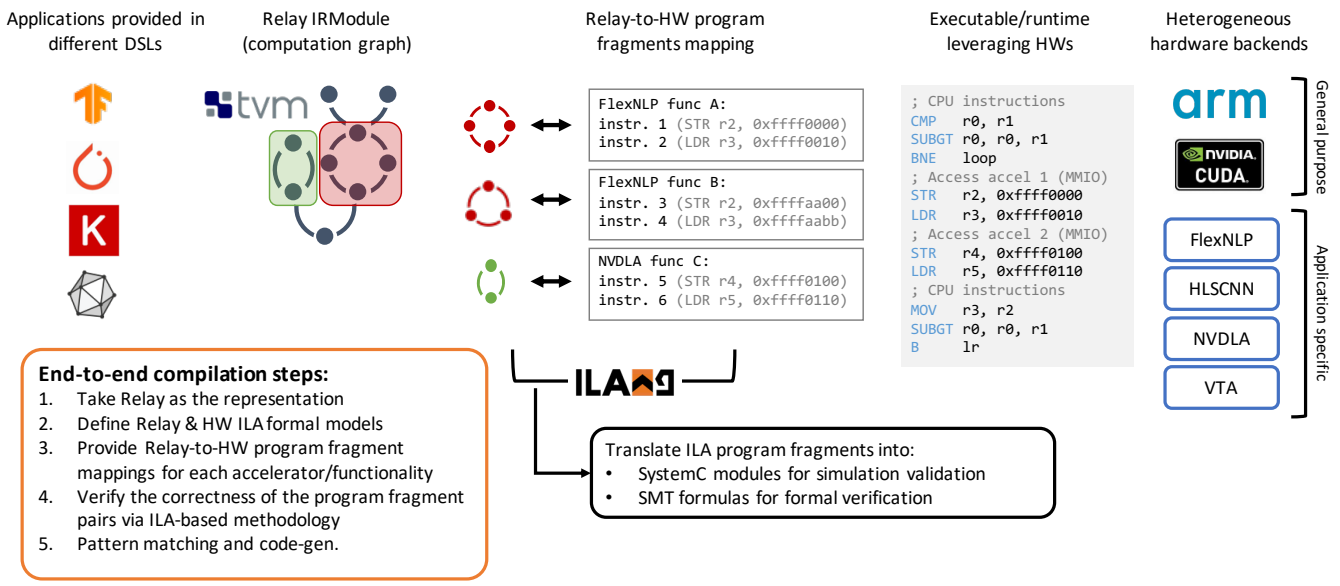

Figure 1: Compilation flow overview

and Python) and verification (using simulation and model checking). ILA's expressiveness and semantics-based approach provide the generality required for a reusable compiler framework with verification support. One tradeoff is that ILA models best support atomic operations; e.g., asynchronous callbacks within an accelerator operation would complicate modeling and verification.

2.3 FlexNLP

FlexNLP is an accelerator optimized for natural language processing (NLP) tasks, with special support for various recurrent neural networks (RNNs) as coarse-grained operations. To boost the accuracy of NLP computations, FlexNLP adopts a custom numerical datatype in its datapath. As RNNs are complex constructs in most DL frameworks but single instructions in FlexNLP, the mismatch in granularity along with the specialized numerics in the hardware present an interesting motivating example for compiler support.

In general, 3LA can also support finer-grained accelerator ISAs like VTA [7]. Granularity tradeoffs in 3LA follow the classic costs and benefits of abstraction: modeling coarser-grained accelerator operations eases pattern matching and verification, but complicates finer-grained optimizations like operator fusion.

2.4 End-to-end compilation

We propose to approach the problem of compiling portions of programs to custom accelerators by finding equivalent mappings between ILA program fragments. In particular, we will define an ILA for Relay, corresponding to portions of its abstract syntax tree (AST), and an ILA for the target accelerator. We can then approach the problem of compilation by identifying sequences of Relay ILA instructions that are equivalent to accelerator ILA instructions, performing a replacement, and lowering the device ILA instructions to the appropriate hardware-software interface (e.g., MMIO commands). Thanks to the formally defined semantics for the ILA, we can verify the equivalence of these sequences through simulation-based and proof-based methods. The simulation-based verification also helps validate the accuracy of custom numeric datatypes.

Implementing a 3LA Code Generator. For our motivating example, we will compile an LSTM speech-to-text application down to FlexNLP, pattern-matching an entire LSTM RNN in Relay without a single manual annotation. We will specify Relay AST fragments (such as particular operators or sequences of of operators) that correspond to accelerator operations, find syntactic matches, and produce the corresponding ILA instructions. The ILA instructions can then be verified and straightforwardly lowered to MMIO operations. We are presently prototyping the proposed compilation scheme using TVM's BYOC framework to handle the syntactic pattern matching. In this way, we will extend the high-level pattern matching capabilities of BYOC with the lower-level reasoning that the ILA enables, creating the foundations for end-to-end verifiable compilation to accelerators.

Conclusion. By providing semantics for the interface between Relay and accelerators, the 3LA methodology supports high-level pattern matching and enables end-to-end reasoning about applications. Further applying semantic reasoning about equivalent program fragments, rather than only syntactic pattern matching, could further increase the degree of automation. The 3LA methodology would not only simplify adding compiler support for custom accelerators but also provide a framework for hardware-software co-design.

In the limit, we hope our uniform, semantics-driven 3LA methodology will enable accelerator designers to simply provide a high-level ILA model of their accelerator, and then apply an ILA-based framework to *automatically* generate reasonable compiler support for unmodified high-level applications, including rewrites for mapping source IR operators to accelerator invocations.

REFERENCES

[1] 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 *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 265–283.

[2] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI'18)*. USENIX Association, USA, 579–594.

[3] Zhi Chen and Cody Yu. 2020. How to Bring Your Own Codegen to TVM. https://tvm.apache.org/2020/07/15/how-to-bring-your-own-codegen-to-tvm

[4] Bo-Yuan Huang, Hongce Zhang, Aarti Gupta, and Sharad Malik. 2019. ILAng: A Modeling and Verification Platform for SoCs Using Instruction-Level Abstractions. In *Tools and Algorithms for the Construction and Analysis of Systems, Proceedings, Part I*. 351–357.

[5] Bo-Yuan Huang, Hongce Zhang, Pramod Subramanyan, Yakir Vizel, Aarti Gupta, and Sharad Malik. 2018. Instruction-Level Abstraction (ILA): A Uniform Specification for System-on-Chip (SoC) Verification. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 24, 1 (2018), 1–24.

[6] 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. *SIGARCH Comput. Archit. News* 45, 2 (June 2017), 1–12. https://doi.org/10.1145/3140659.3080246

[7] Thierry Moreau, Tianqi Chen, Luis Vega, Jared Roesch, Eddie Yan, Lianmin Zheng, Josh Fromm, Ziheng Jiang, Luis Ceze, Carlos Guestrin, et al. 2019. A Hardware-Software Blueprint for Flexible Deep Learning Specialization. *IEEE Micro* 39, 5 (2019), 8–16. https://doi.org/10.1109/MM.2019.2928962

[8] Jared Roesch, Steven Lyubomirsky, Marisa Kirisame, Logan Weber, Josh Pollock, Luis Vega, Ziheng Jiang, Tianqi Chen, Thierry Moreau, and Zachary Tatlock. 2019. Relay: A High-Level Compiler for Deep Learning. arXiv:1904.08368 [cs.LG]

[9] Thierry Tambe, En-Yu Yang, Zishen Wan, Yuntian Deng, Vijay Janapa Reddi, Alexander Rush, David Brooks, and Gu-Yeon Wei. 2020. Algorithm-Hardware Co-Design of Adaptive Floating-Point Encodings for Resilient Deep Learning Inference. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference* (Virtual Event, USA) *(DAC '20)*. IEEE Press, USA, Article 51, 6 pages.