SAND2021-0995R

Using MLIR Framework for Codesign of ML Architectures, Algorithms and Simulation Tools

Cannada Lewis,[1] Clay Hughes,[1] Simon Hammond,[1] and Sivasankaran Rajamanickam[1]

[1] *Sandia National Laboratories*[a], *Albuquerque, NM 87185, USA*

I. MLIR FRAMEWORK

MLIR (Multi-Level Intermediate Representation) [5], is an extensible compiler framework that supports high-level data structures and operation constructs. These higher-level code representations are particularly applicable to the artificial intelligence and machine learning (AI/ML) domain, allowing developers to more easily support upcoming heterogeneous AI/ML accelerators and develop flexible domain specific compilers/frameworks with higher-level intermediate representations (IRs) and advanced compiler optimizations. The result of using MLIR within the LLVM compiler framework [4] is expected to yield significant improvement in the quality of generated machine code, which in turn will result in improved performance and hardware efficiency.

The multi-level nature of MLIR allows for a hierarchical representation of the code within the LLVM compiler infrastructure, with different levels referred to as *dialects*. A simple example of this is a linear algebra code that is compiled to a high level MLIR representation using matrix or linear algebra dialects. At the high level dialect, it is possible to make transformations that depend on knowledge and properties of matrix operations. For example, the operation $\mathbf{A} = (\mathbf{M}^\mathbf{T})^\mathbf{T}$, which assigns the transpose of the transpose of $\mathbf{M}$ to $\mathbf{A}$, can be rewritten as $\mathbf{A} = \mathbf{M}$. Such matrix operations can then be converted into a lower level dialect of MLIR, a process referred to as *lowering*, such as a loop dialect. Eventually the representation will be lowered to a dialect that has the ability to generate machine code - whether for a contemporary system, custom hardware, or even a hardware description for an FPGA. This approach provides a powerful, extensible compiler framework supporting efficient tensor/matrix operations for AI/ML applications.

The most powerful feature of MLIR is the ability to design dialects that represent code at a level that is conducive to applying transformation and optimizations that would be difficult or impossible to achieve with a single general purpose IR. In the above transpose example, if the code was compiled directly to a low level representation that obscured the origin and special structure of the input, it would have been all but impossible to detect the simplifying transformation that is obvious at a matrix level. Similarly, the matrix level doesn't need to be concerned with details such as common loop optimizations (e.g. loop tiling and unrolling) or hardware-specific transformations. MLIR simplifies the process of holistic program inspection and optimization by allowing the compiler developer to leverage the appropriate abstractions at the appropriate times in the lowering process.

Figure 1 shows some possible paths in an MLIR compiler for a code written in high-level language (*e.g.* Python, TensorFlow) to be compiled to either FPGA, PTX (for GPUs) or X86_64 (for CPUs).

II. ARCHITECTURE VENDOR SUPPORT OF MLIR

Although originally developed at Google, MLIR has been released as open source as part of the LLVM open compiler framework [1] enabling its use in a wide variety of communities. Partners in the project include AMD, ARM, Cerebras, Google, Graphcore, Habana, Intel, Mediatek, NVIDIA, Qualcomm, SambaNova, Samsung, Xiaomi and Xilinx. Google has focused primarily on using MLIR within TensorFlow [2], especially for their TPU machine learning systems. Other vendors are starting to support MLIR because of: (1) its flexibility and extensibility; (2) its popularity and the view that there is a critical mass behind its use, and, (3) programming models such as TensorFlow, PyTorch, Jax, and others already provide a path to generate MLIR, lowering the amount of work the vendors must do to target existing software.

---

[a] Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

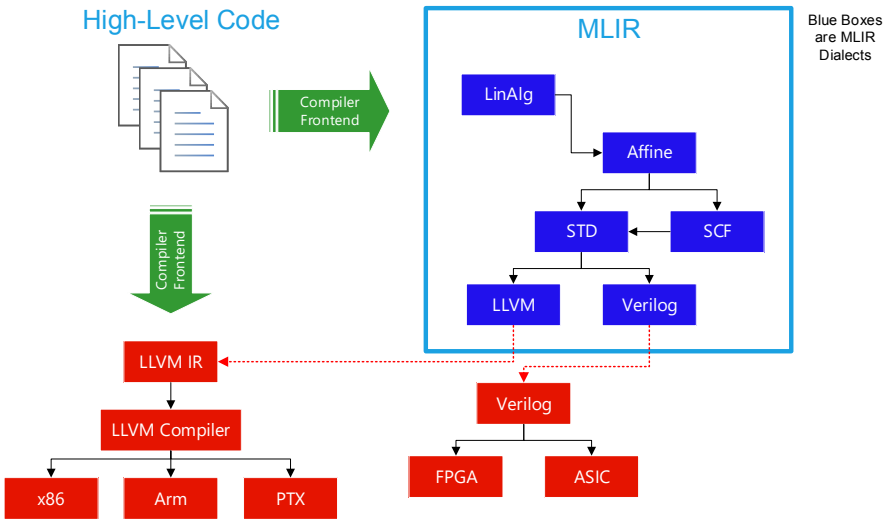

Figure 1. Diagram of some possible paths through the MLIR compiler. Blue boxes are different MLIR dialects.

III. POTENTIAL USE OF MLIR FOR ALGORITHM DEVELOPMENT

There are several approaches where the MLIR framework could be utilized as we codesign algorithms and future hardware architectures. Several emerging machine learning architectures (Cerebras, Samba Nova, NextSilicon, GraphCore) are based on the spatial/data flow paradigm. By effectively utilizing data flow between processing units, these accelerators provide significant reductions in memory movement, which can improve performance and energy efficiency. However, in contrast to GPUs/CPUs, there are no easy methods to program these emerging data flow architectures, at least outside of vendor-specific frameworks. Designing an MLIR-based intermediate representation for expressing data flow algorithms would allow code or algorithm developers to develop new algorithms that are based on the principles of data flow, while maintaining hardware agnostic programming languages. Such an MLIR dialect could be lowered to code that targets multiple hardware targets or to an IR which would be amenable to hardware codesign capabilities via simulation using tools such as the SST Hardware Simulation Framework [6]. As part of the ARIAA co-design center (ASCR funded), we are already engaged in developing a dialect for expressing data flow algorithms. Once we have the capabilities to develop algorithms, we can evaluate data flow algorithms for important kernels from NNSA codes such as Kokkos Kernels and Trilinos solvers.

IV. POTENTIAL USE OF MLIR IN SIMULATION FOR ASCR AND NNSA

We are leveraging analytical modeling tools for dataflow accelerators and have developed dataflow components in SST as part of the ASCR ARIAA project. Our strategy to target these dataflow components utilizes MLIR dialects developed within the ARIAA project to compile optimized algorithm implementations for low-level hardware formats [3]. In addition, the Advanced Machine Learning part of Next Generation Computing Technologies element in CSSE portion of the ASC program is funding the development of analytical modeling tools that target neuromorphic and data flow accelerators, which may also leverage the MLIR framework. The combination of MLIR, the flexible design capabilities of SST, and the speed of analytical modeling tools will result in fast codesign cycles for both ASCR and NNSA studies, opening up opportunities to evaluate future hardware capabilities far ahead of their availability as a product. Furthermore, the knowledge obtained from these studies will allow ASCR and NNSA laboratories to more precisely evaluate and communicate specific requirements to vendors for the next-generation of high-performance computing and ML/AI based systems.

---

[1] Multi-level intermediate representation overview. https://mlir.llvm.org. Accessed: January, 2021.

[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, Dandelion 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. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Clayton Hughes, Simon David Hammond, and Robert J. Hoekstra. CDFG Extraction Tool for LLVM. Internal Report SAND2020-0937, 2020.

[4] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *International Symposium on Code Generation and Optimization, 2004 (CGO 2004)*, pages 75–86. IEEE, 2004.

[5] Chris Lattner, Jacques Pienaar, Mehdi Amini, Uday Bondhugula, River Riddle, Albert Cohen, Tatiana Shpeisman, Andy Davis, Nicolas Vasilache, and Oleksandr Zinenko. MLIR: A Compiler Infrastructure for the End of Moore's Law. *arXiv preprint arXiv:2002.11054*, 2020.

[6] Arun F Rodrigues, K Scott Hemmert, Brian W Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, Rolf Risen, Jeanine Cook, Paul Rosenfeld, Elliot Cooper-Balis, et al. The Structural Simulation Toolkit. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):37–42, 2011.