Compiler Engineering
CS6383 - Sep 2019
Mini Assignment 3:

MLIR and LLVM

INSTRUCTOR: Dr. Ramakrishna Upadrasta

Report By:
Vijay Tadikamalla - CS17BTECH11040
Tungadri Mandal - CS17BTECH11043

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

MLIR as a LLVM Subproject

The original aim of the MLIR project was to define an **Intermediate Representation(IR)** that will unify the infrastructure required to execute high performance machine learning models in **Machine Learning frameworks**. It was supposed to reduce the cost to bring up new hardware. Aside from being state of the art, the implementation was not specific to these applications and it was also being adopted by **Flang compiler.** (More about the Flang compiler is explained in the next topic).
So, on discussion with a large number of *'industry partners'* the MLIR and LLVM team decided to build a shared ML compiler infrastructure under a common umbrella with well neutral governance. So, the MLIR team proposed that **MLIR core** join the non profit LLVM foundation as a new subproject. MLIR is open source on GitHub and MLIR core is located in github/tensorflow/mlir, '*including an application independent IR, the code generation infrastructure, common graph transformation infrastructure, declarative operation definition and rewrite infrastructure, polyhedral transformations etc.*' They also claimed that they plan to follow the **LLVM developer policy** which defines the project's policy towards developers and their contributions and that they have also been following LLVM-style development process from the beginning - including all relevant coding and testing styles and that they build on core LLVM infrastructure pervasively.

MLIR at its core is a flexible infrastructure for modern optimizing compilers. This means it consists of a specification for intermediate representation(IR) and a code toolkit to perform transformations on that representation. It was already highly influenced by LLVM and unabashedly reused many great ideas from it.

To separate different hardware and software targets, MLIR has ***"dialects"*** including **LLVM IR dialect**, which has 1:1 mapping between it and LLVM's own representation, allowing MLIR to emit GPU and CPU code through LLVM. This dialect wraps the LLVM IR types and instructions into MLIR types and operations. It provides several additional operations that are necessary to cover for the differences in the IR structure.

So, as MLIR already uses LLVM and LLVM IR is one if the MLIR dialects, and furthermore MLIR is a compiler infrastructure, it fits as a natural part of **LLVM ecosystem**. MLIR also provides as a base to support frontend-specific IRs transformations. For example, current development of the new Flang compiler depends on MLIR. Flang is becoming a subproject of LLVM and MLIR should be a part of LLVM. MLIR also has some unique capabilities such as the analysis of multi-dimensional

loops, that can be moved into LLVM and used by both LLVM- and MLIR-level transformations.
Another advantage of this move as mentioned was that MLIR could easily become a simpler entry point into LLVM from other languages, frameworks and optimisation plugins. A more abstract representation and a more stable IR generation from it also has the possibility to make maintenance of external projects much easier. This would benefit research as much as enterprise and by consequence the LLVM project.

An example of something that MLIR is trying to solve elegantly on top of LLVM is helping with **heterogeneous computing**. As mentioned : *"Today a compiler framework that would try to support a device accelerator (like a GPU) would need to manage outside of / above LLVM how to split the host and device computation. MLIR allows to have both in the same module, and providing some convenient facility for the "codegen" and integration with LLVM."*

MLIR is very extensible and it lets the user define their own abstraction and compose on top of whatever the community will want to propose in the core. But this decision of having MLIR as a LLVM subproject also has many **implications** which may or may not be negative.

As mentioned it can be **confusing** for a frontend developer looking to use LLVM backend technology and a developer looking to implement different kinds of functionality to choose whether to target or enhance MLIR components, LLVM components, or both. As one user puts it *" I'm also worried that it's too broad in respect to what it can do on paper, versus what LLVM can handle on code."*

It is mentioned by people that MLIR and LLVM IR should interact within known boundaries and expected behaviour. The person who suggested this claims that this is not to say that MLIR can't be used for anything else but just to say that *" by being inside the repo, and maintained by our community, LLVM IR would end up as the *primary* target, and there will be a minimum stability/functionality requirements."*

Implications of MLIR for FORTRAN

We all know that FORTRAN is the **world's first high-level programming language**.
FORTRAN was initially designed to help scientists and engineers to translate mathematics into simple, readable, and fast code. So, it has features like rich array operations, complex numbers, exponentiation, special functions, etc.
FORTRAN is more restrictive than other languages like C or C++, so the resulting code is easier to maintain and easier for compilers to optimize.

Compilers of FORTRAN

Flang is a Fortran language front-end designed for integration with LLVM and the LLVM optimizer. While the Flang front-end in its current form is already successfully in use, there's still lots of room for improvement. One of the potential improvement is use of MLIR.

MLIR can be used to redefine and implement a high-level operational representation of Fortran because:

1. MLIR focuses on machine learning infrastructure, high performance accelerators, heterogeneous compute, and HPC-style computations.
2. It also supports a refined polyhedral model for transforming affine operations on array data.

These mathematically-based representations will help realize FORTRAN compiler goals of composing, transforming, and optimizing array operations for HPC.

Current Progress

A **new front-end for Fortran** was announced at **EuroLLVM April 17, 2018**.
F18 is a new Fortan front-end which has been approved to become the standard Fortran compiler for LLVM and is in the process of being adopted into LLVM proper. By the beginning of 2020, F18 is expected to replace Flang as a complete implementation of Fortran 2018.

FIR

Currently, MLIR has no support for Fortran out of the box. So, the support for Fortran must be built and added through the extensibility features of MLIR.
This process of extending MLIR is called adding a dialect. So, **FIR dialect** is a well-defined set of operations, types, etc. that captures the executable semantics and state of a correctly specified Fortran program.

Due to all the above mentioned reasons, there is a huge potential to use MLIR as a substrate upon which to build an FIR dialect.

References

1. https://groups.google.com/forum/#!topic/llvm-dev/_9dr6ZvWWAQ/discussion
2. https://lfortran.org/
3. http://llvm.org/devmtg/2019-10/talk-abstracts.html#tech19
4. https://github.com/flang-compiler/f18/blob/master/documentation/Investigating-FIR-as-an-MLIR-dialect.md
5. http://fortranwiki.org/fortran/show/Flang
6. https://github.com/flang-compiler/flang/wiki