Federal University of Minas Gerais
Departament of Computer Science
Mentor: Nicholas Wilson
Student: Roberto Gomes Rosmaninho Neto

Symmetry Autumn of Code 2019
Compiler's Laboratory

Multi-Level Intermediate Representation Support for LLVM-Based D Compiler [1]

The Multi-Level Intermediate Representation(MLIR) is a project that aims to define a flexible and extensible intermediate representation(IR) that will unify the infrastructure required to execute high performance machine learning models in TensorFlow and similar ML frameworks. To achieve its goals, MLIR introduces notions from the polyhedral[2] domain as first class concepts and the notion of dialects. These dialects consist in a set of defined operations that makes the levels of optimizations more flexible than the usual source code, LLVM IR and machine code. Each dialect can be visualized as a new level of IR. Each new level enables some optimizations that may not be available at other levels. GPU[3] is one of the Dialects available in MLIR today, this dialect provides middle-level abstractions for launching GPU kernels following a programming model similar to that of CUDA or OpenCL. Some other examples of dialects are Affine[4], Vector[5] and LLVM IR[6].

The LLVM-Based Compiler(LDC) is a project that aims to create a portable D programming language compiler with modern optimization and code generation capabilities[7].LLVM and the LLVM IR have been used by D programmers mainly for lower-level optimizations and to implement some program analyses. However, the LLVM IR is closer to machine code rather than source code. This fact makes it difficult to rely on the LLVM IR to carry out language-specific optimizations. Thus, given the support of high level optimizations that MLIR provides, a new level of abstraction can be created for D. This level can be seen as a middle ground between the source code and the LLVM IR.

The goal of this project is to provide LDC with a new level of abstraction to support the integration of the MLIR into the D ecosystem. This project will lead to the incorporation of new optimizations into the D programming language. The addition of new optimizations will be possible due to the levels of intermediate representation provided by the MLIR infrastructure. Ultimately, we shall generate XLA for use with TensorFlow. Thus, although an MLIR representation will be useful specifically for Machine Learning programs written in D, other programs may also benefit from the MLIR features to gain performance due the polyhedral concept implemented by MLIR to use some analysis and optmizations such as loop optimizations across kernels (fusion, loop interchange, tiling, etc) and high-performance in matrix multiplication. To complete these tasks, this project aims to create a D Dialect of MLIR, which will be closer to the D source code. Consequently, we will be able to use language specific optimizations such as GC2Stack[8] in D at

[1] `https://github.com/dlang/projects/issues/2`
[2] `https://polly.llvm.org`
[3] `https://github.com/tensorflow/mlir/blob/master/g3doc/Dialects/GPU.md`
[4] `https://github.com/tensorflow/mlir/blob/master/g3doc/Dialects/Affine.md`
[5] `https://github.com/tensorflow/mlir/blob/master/g3doc/Dialects/Vector.md`
[6] `https://github.com/tensorflow/mlir/blob/master/g3doc/Dialects/LLVM.md`
[7] `https://wiki.dlang.org/LDC`
[8] `https://github.com/ldc-developers/ldc/blob/master/gen/passes/GarbageCollect2Stack.cpp`

dialect level. We shall ensure the necessary infrastructure to port this D dialect of MLIR to other dialects, so that more optimizations might be available to improve the runtime of D programs. This portabilitiy will

I am motivated to work on this project due to a number of reasons. First, I have great familiarity with LLVM, having used it in a previous project, which, actually, involved the D programming language. The goal of that project was to design and implement a static program analysis to pinpoint function arguments that could have been made lazy. Second, I work in a compilers lab, having experience with compiler construction, and contact with compiler engineers. Third, I am very familiar with the C++ programming language, having worked in several projects whose main implementation was using it. If this project is accepted, it will give me material for my undergraduate research project; thus, even after its formal conclusion, I am still planing to dedicate 20 hours every week to work on its improvement, or on some topic related to the interface between D and the LLVM compilation infrastructure.