SPIR-V to LLVM IR dialect conversion in MLIR

Student: George Mitenkov
Mentors: Lei Zhang, Mahesh Ravishankar

1 Introduction

MLIR is a novel compiler infrastructure that enables multi-level abstraction and therefore enhances reusability and extensibility. The core concept that allows to extend MLIR is a dialect. It is an abstraction level with its own types, attributes and operations that is designed to solve lowering, optimization problems, etc.

In MLIR, compilation happens via dialect conversion, in a progressive and principled approach. Each dialect can be partially or fully converted from one to another, preserving the semantic meaning of the program. There already exist dialects that model SPIR-V and LLVM IR, however there is no conversion between them at the moment. In my project, I propose to develop and test a conversion path that lowers SPIR-V to LLVM IR.

2 Benefits

Having a SPIR-V to LLVM IR dialect conversion within MLIR embraces SPIR-V dialect into the greater LLVM ecosystem: we will be able to convert SPIR-V into LLVM IR dialect and then export it to LLVM proper. Such a conversion path has great benefits including but not limited to generating CPU machine code for SPIR-V and JITing SPIR-V. This would allow to support SwiftShader, a CPU-based Vulkan implementation, as well as LLVM-based GPU hardware driver compilers such as AMDVLK.

3 The project

SPIR-V serves multiple execution environments. In my project, I focus particularly on the case of Vulkan compute.

For the project, my plan is to start with the conversion of simple operations (*e.g.* basic arithmetic/logical operations) and types (scalars), `spv.func` and `spv.module`. Then, scale up to support control flow, and also deal with SPIR-V's challenges such as entry points or specialization constants. Lastly, I propose to add support for SPIR-V's GLSL extended instruction set. The full breakdown within the project's phases is described in the Deliverables section. In addition, I want to develop a tool for correctness verification. This can be a stretch goal and is also described further.

Throughout the project, there are several important tasks I have to do in order that my project develops smoothly.

3.1 SPIR-V to LLVM mapping

Firstly, it is important to compare SPIR-V dialect semantics with LLVM IR dialect semantics. Since the result of conversion must preserve the same semantic meaning, it is extremely important to define correct conversion patterns.

Most SPIR-V dialect operations have a one-to-one mapping to LLVM IR dialect operations (*e.g.* `spv.IAdd` → `llvm.add`). However, there exist some edge cases that have to be considered with caution separately. A good example of such a case is `OpUndef` instruction which has non-deterministic behaviour and does not have a well-defined equivalent in LLVM. Also, some SPIR-V operations cannot be mapped

directly into LLVM operations, and need to be converted to LLVM intrinsics (*e.g.* `spv.Transpose` → `llvm.matrix.transpose.*`). This specifically applies to SPIR-V's extended instruction set that specifies `GLSL.std.450`, where nearly all operations correspond to LLVM's intrinsics.

3.2 Conversion

The second task is to define the conversion, which includes the following components.

- Dialect conversion structure
  The conversion will be in `mlir/lib/Conversion/SPIRVToLLVM/` directory. It will contain:
  - `CMakeLists.txt` file
  - `SPIRVToLLVMPass` to specify a pass for the conversion
  - `SPIRVToLLVM` file where conversion of SPIR-V ops to LLVM IR ops is implemented

  Running `mlir-opt -convert-spirv-to-llvm <filename.mlir>` will invoke a conversion pass.
- Type conversion
  Handled by `TypeConverter` subclass, specifying how to map one type to another, *e.g.* `spv.i1` → `llvm.i1`.
- Operation conversion
  Handled by `ConversionPattern` subclasses.
- Tests
  Testing makes use of FileCheck with LLVM Integrated Tester. This allows running 'diff' tests. The tests will be placed in `mlir/test/Conversion/SPIRVToLLVM/`. If implemented, `mlir-spirv-runner` would be placed in `mlir/tools/mlir-spirv-runner/` directory.

3.3 MLIR-SPIRV-runner

For testing the conversion, I would first use the FileCheck tests (as for other conversions on GitHub in `llvm-project/mlir/test/Conversion/`). However, I think that it is also important to check the overall validity of the conversion and its performance. Particularly, I want to compare the result of running via SPIR-V to LLVM conversion with the result of the execution of MLIR on Vulkan. The latter is already implemented through `mlir-vulkan-runner`. Therefore, for correctness verification a tool to run SPIR-V (that uses SPIR-V dialect to LLVM IR conversion) is needed. Consequently, I propose to develop a `mlir-spirv-runner`. Since this would mean extra work in the coding period, and also given its complexity, I think that it can be considered as a stretch goal.

4 Deliverables

During the whole project I intend to keep a clear documentation and a list with bugs, TODO features, etc. I also have an idea of writing a simple script prior to the project that will indicate the percentage of the currently supported conversions. This will help to keep track of the status of the project.

Community Bonding Period: 4th May – 31th May: During this period, I will get to know the community better and deepen my understanding of MLIR in general. I will also research more information about SPIR-V (both through specification and the dialect implementation on GitHub). Moreover, I will investigate how the current conversions are implemented: *e.g.* look at conversion documentation from standard to LLVM.

First coding phase: 1st June – 3rd July: During the first phase of the project I want to focus on getting the minimum viable product. I think it can be a conversion of a `spv.module` that contains `spv.func` with some basic operations (*e.g.* `spv.IMul`).

- weeks 1-3: Have a conversion running end-to-end for some simple binary operator (*e.g.* `spv.IAdd`). Test the conversion through FileCheck. I think some time should be left for familiarity and code review issues.
- weeks 4-5: Completing MVP: handling `spv.module`, `spv.func`, and some other simple operations (arithmetic/logical/bit operations). Again, have FileCheck tests for the operations introduced.

Second coding phase: 4th June – 31th July: During the second phase, I propose to scale across `spv.loop`, `spv.selection` and other control flow operations, as well as to add support for more types (`spv.array`, `spv.rtarray`, `spv.pointer` and some others). Also, I plan to solve some SPIR-V specific challenges (*e.g.* entry points, specialization constants, etc.).

- weeks 6-7: Scale across control flow operations and other types. Add FileCheck tests for each new type and operation.
- weeks 8-9: Work on entry points and specialization constants. This may be continued in the final phase.

Final coding phase: 1st August – 31th August: At this stage I plan to focus on SPIR-V's GLSL extended instruction set. Additionally, I may want to work on my stretch goals. This particularly involves implementing a `mlir-spirv-runner` (see MLIR-SPIRV-runner subsection).

- weeks 10-13: Implement features from extended instruction set (`spv.GLSL.Exp`, `spv.GLSL.Sin`, `spv.GLSL.Cos` and others), have FileCheck tests covering those.
- week 14: Dedicate the last week to wrap up and submit all pending work.
- stretch goal: Develop a `mlir-spirv-runner`. Verify the correctness of conversion by comparing the results with execution of `mlir-vulkan-runner`.

5 About me

I am a 2nd year student, studying BEng Mathematics and Computer Science at Imperial College London. I am interested in Optimization and Compilers. At university, I have successfully completed various big courseworks (details can be found below) that taught me how to work with a huge codebase, collaborate effectively via git, and develop the software in a TDD manner. Last summer I was an intern at Sberbank. My job was to provide a stable backend and optimize the time when requesting the data from the database (via both SQL syntax and use of suitable data structures and algorithms).

Skillset

Python	- WACC language to ARM compiler
C	- Pintos OS
	- ARM Assembler
Java	- SpringBoot backend for Sberbank's web service

Contact details
Mail: georgemitenk0v@gmail.com

6 References

1. LLVM IR semantics from http://llvm.org/docs/LangRef.html
2. SPIR-V specification from https://www.khronos.org/spir/