Compiler Design Notes

Ayush Raina

March 25, 2025

Multi Level Intermediate Representation (MLIR)

MLIR stands for multi level intermediate representation, it is a flexible and extensible compiler under the LLVM Project. It allows representing and optimizing code across multiple abstraction levels. It supports:

1. **Multiple Dialects:** Custom IR for different domains (e.g Tensorflow, PyTorch etc)
2. **Progressive Lowering:** Transforming high level constructs to lower-level forms.
3. **Reusability:** Shared Optimizations across different compiler pipelines.

Question. What do we mean by multiple abstraction levels ?
In MLIR, multiple abstraction levels mean representing code at different stages of computation ranging from high level domain specific applications like math operations to low level machine instructions.

- **High Level Abstraction (Tensor Dialect):** In this level of abstraction, we represent code in a domain specific way. For example, operations like matrix multiplication, convolution look like how we write them in Tensorflow or PyTorch.

```
%result = "linalg.matmul"(%A, %B) : (tensor<2x3xf32>, tensor<3x2xf32>) -> tensor<2x2xf32>
```

- **Mid Level Abstraction (Affine Dialect):** In this level of abstraction, we can see the linear algebra transformations and operations like matmul converted to a loop nest.

```
affine.for %i = 0 to 2 {
    affine.for %j = 0 to 2 {
        affine.for %k = 0 to 3 {
            %c = arith.mulf %A[%i, %k], %B[%k, %j]
            %sum = arith.addf %sum, %c
        }
    }
}
```

- **Low Level Abstraction (LLVM Dialect):** In this level of abstraction, we see machine level instruction which directly operates on pointers and raw data like load, store, getelementptr etc as shown below.

```
%ptr = llvm.getelementptr %A[%i, %k]
%val = llvm.load %ptr
```

MLIR is designed to support progressive lowering (Tensor $\rightarrow$ Affine $\rightarrow$ LLVM) and reusability of optimizations across different compiler pipelines, for example - Loop Unrolling is useful for both HPC and Tensorflow programs.

Core Structure of MLIR (Hierarchy)

1. **Operation (Op):** The smallest unit of computation in MLIR. In the example shown below **arith.addf** is the operation which adds two floating point numbers.

```
%sum = arith.addf %a, %b : f32
```

2. **Value:** Data produced or consumed by operations in SSA Form. In the above example **%sum, %a, %b** are values.
3. **Block:** A sequence of operations (similar to basic block in LLVM). Block also defines control flow by having terminators like **return** etc.

4. **Region:** A container of blocks, enabling nested control flow. For example loops or functions. An **affine.for** has region for loop contents.

5. **Attribute:** Immutable metadata attached to operations. For example constants, symbolic info etc.

```
%cst = arith.constant 10 : i32 // 10 is an attribute
```

6. **Type:** Describes the kind of data a value holds. For example **f32, tensor**$< 4 \times f32 >$ etc.