编译原理大作业Tutorial

助教：张炜创，黄世远

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录 Contents

1 MLIR 编译框架简介

2 Tiny 语言简介

3 大作业问题描述

4 实验流程及演示

目录 Contents

1 MLIR 编译框架简介

2 Tiny 语言简介

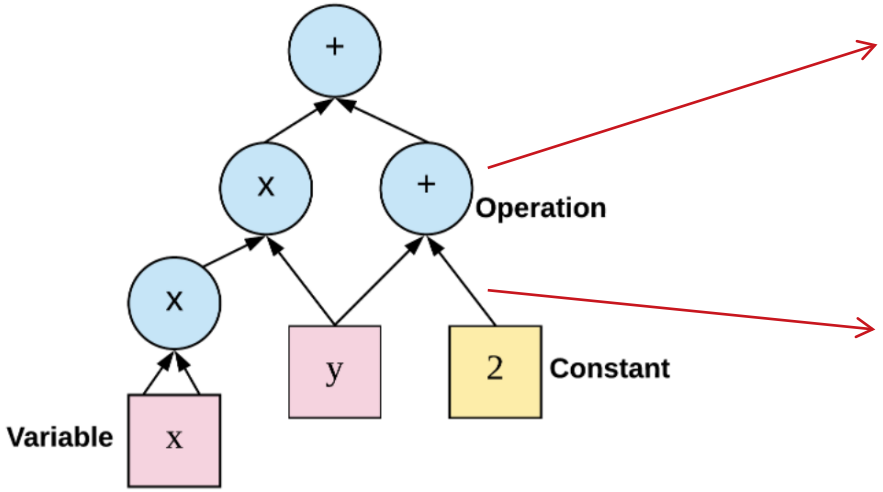
3 大作业问题描述

4 实验流程及演示

以TensorFlow为引

TensorFlow 是一个端到端开源机器学习平台

在TensorFlow中，使用计算图 (computational graph) 来表示计算任务。



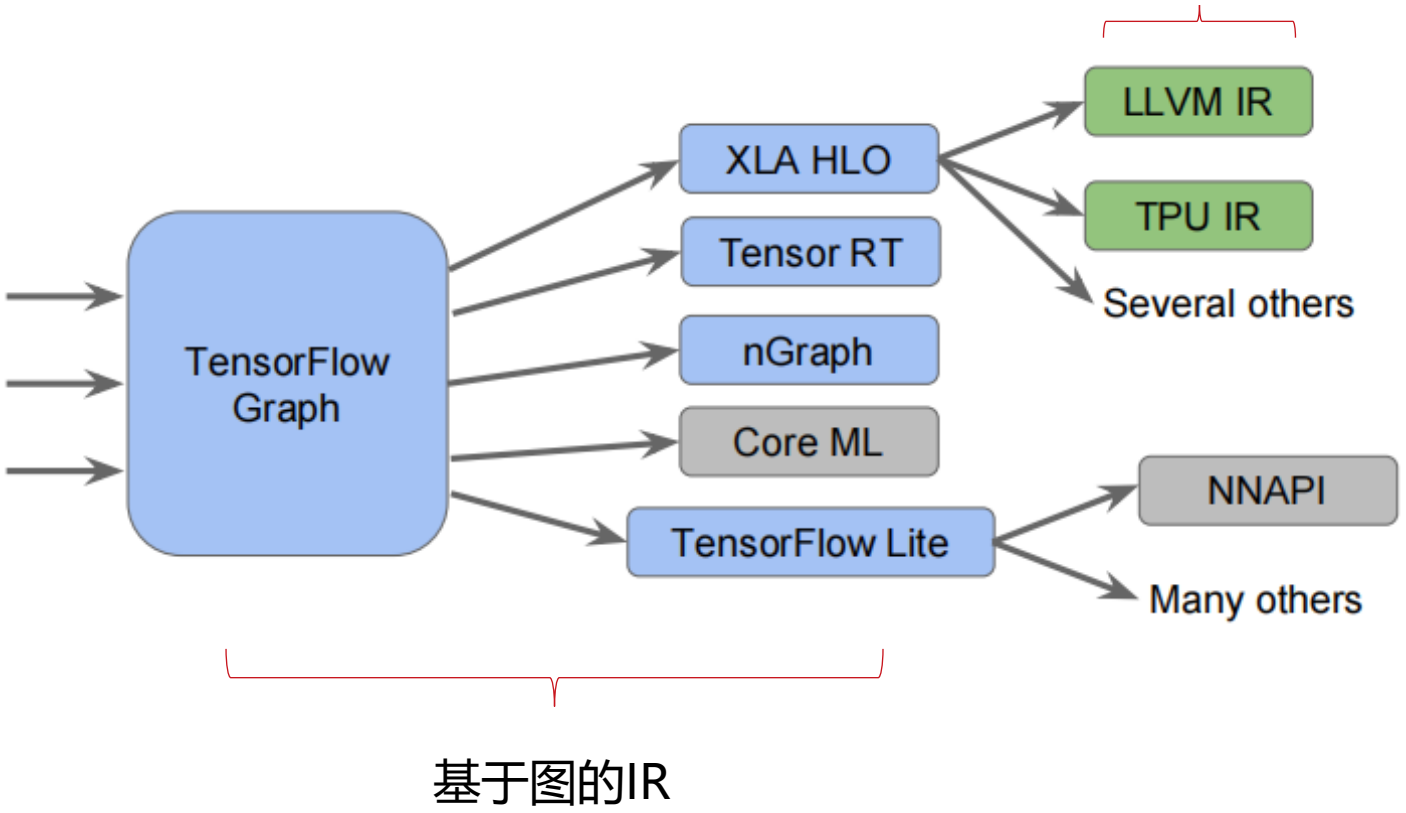
节点表示具体操作，比如multiplication，add等

边表示数据流

函数f(x,y)=x^2y+y+2对应的计算图

以TensorFlow为引

TensorFlow生态：



存在问题：

- 不同IR间的抽象级别差距太大
- 不同层次内部的优化手段无法复用/迁移
- 各部分基础设施大量重复（重复造轮子）

什么是MLIR？

MLIR (Multi-Level Intermediate Representation)

不是machine learning， 但machine learning是其应用领域之一

The MLIR project is a novel approach to building reusable and extensible compiler infrastructure. MLIR aims to address software fragmentation, improve compilation for heterogeneous hardware, significantly reduce the cost of building domain specific compilers, and aid in connecting existing compilers together.

一个可重用、可扩展的开源编译框架

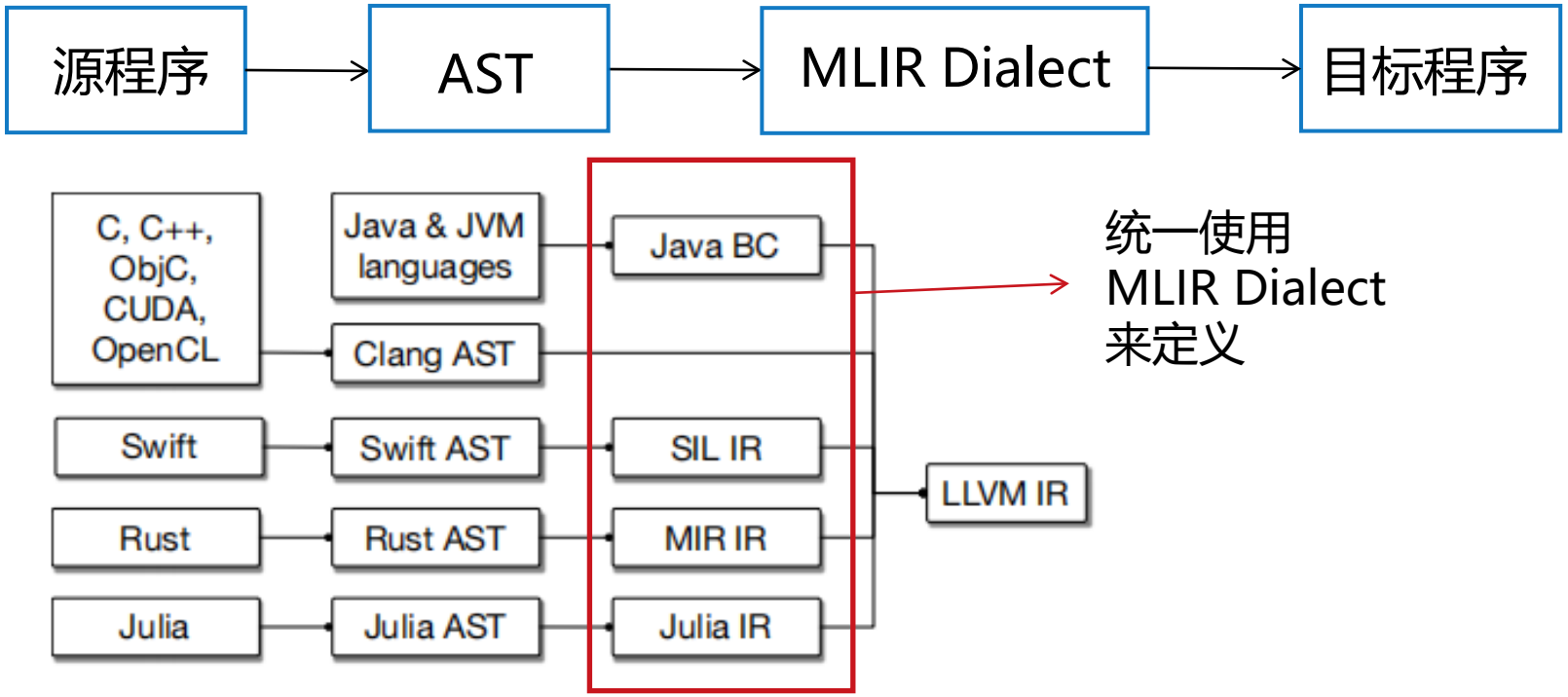
- 解决软件碎片化问题
- 为异构硬件提供编译支持
- 降低领域专用编译器的构建难度
- 将现有的优秀编译器工作连接在一起

MLIR

MLIR的核心概念：Dialect

针对前面TensorFlow中提到的一系列问题，有什么比较好的解决方案?

统一IR表示！！！

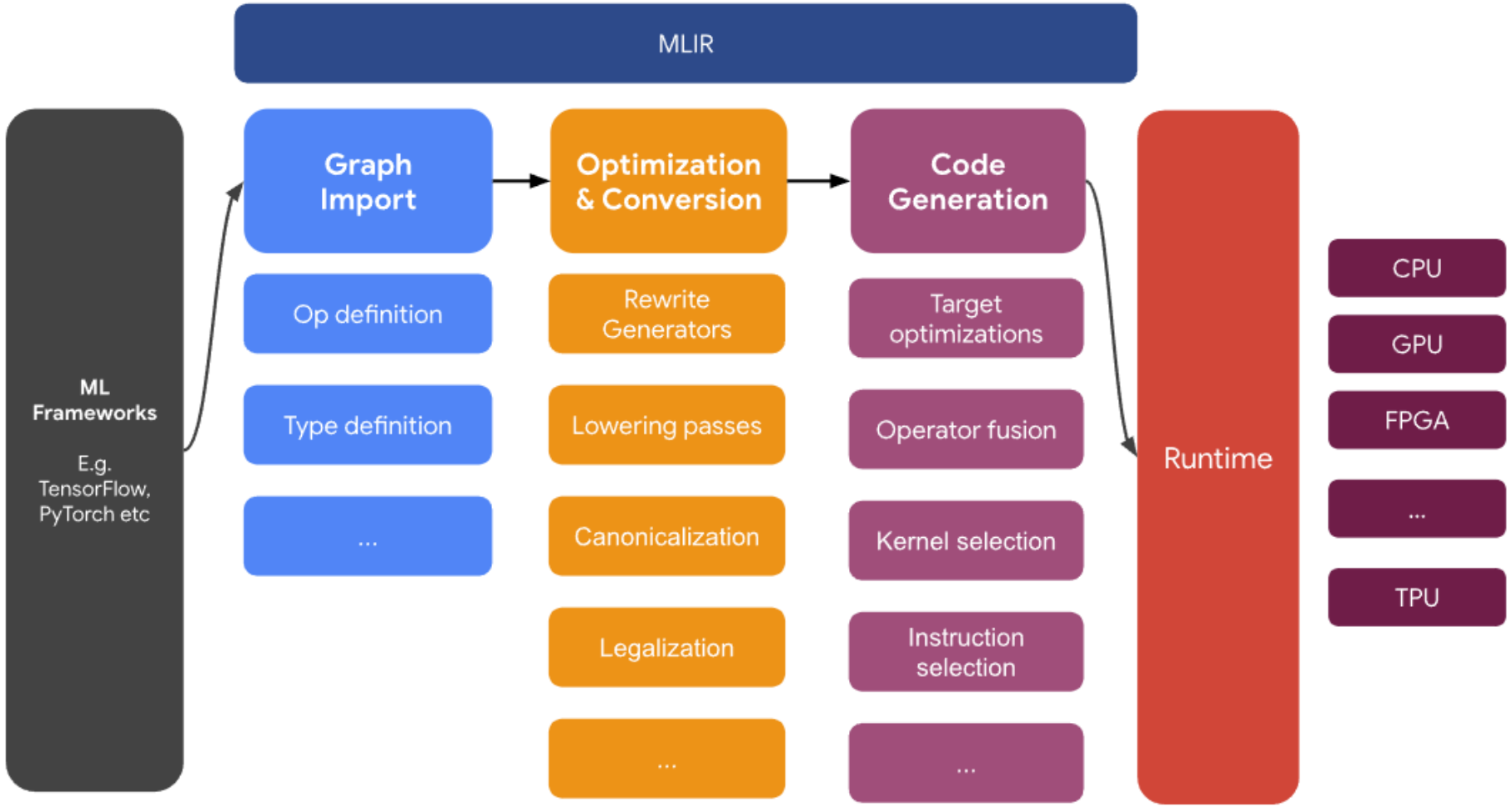


Dialects Docs

- 'acc' Dialect
- 'affine' Dialect
- 'amx' Dialect
- 'arith' Dialect
- 'arm_neon' Dialect
- 'arm_sve' Dialect
- 'async' Dialect
- 'bufferization' Dialect
- 'cf' Dialect
- 'complex' Dialect
- 'dlti' Dialect
- 'emitc' Dialect
- 'func' Dialect
- 'gpu' Dialect
- 'linalg' Dialect
- 'llvm' Dialect
- 'math' Dialect
- 'memref' Dialect

MLIR

以TensorFlow为例



目录 Contents

1 MLIR 编译框架简介

2 Tiny 语言简介

3 大作业问题描述

4 实验流程及演示

Tiny 语言简介

Tiny语言是本次大作业的使用语言。它是一种自定义的，基于张量（tensor）的语言。Tiny语言支持少量功能，包括函数定义，基本数学运算以及打印功能。

定义转置相乘的函数

```
def multiply_transpose(a, b) {
  return transpose(a) * transpose(b);
}
```

张量对应元素相乘

main函数

Reshape操作

```
def main() {
  var a<2, 3> = [[1, 2, 3], [4, 5, 6]];
  var b<2, 3> = [1, 2, 3, 4, 5, 6];
  var c = multiply_transpose(a, b);
  var d = multiply_transpose(b, a);
  print(d);
}
```

打印计算结果

MLIR

一个端到端流程

词法&语法分析

Lowering

在遍历AST的过程中，根据AST节点的信息，使用dialect定义相应操作，实现AST节点到MLIR dialect operation的绑定

例如：

lower 到 affine dialect，进行循环优化

lower 到 LLVM dialect，方便下一步输出LLVM IR

MLIR

目录 Contents

1 MLIR 编译框架简介

2 Tiny 语言简介

3 大作业问题描述

4 实验流程及演示

大作业简介

作业主要内容（20分）

- 基础部分（14分）：考察词法与语法分析知识，要求补全Tiny语言的词法与语法分析代码，并对“var”的表示引入新的扩展。
- 进阶部分（6分）：对代码优化问题进行初步探索，要求对Tiny语言的transpose()函数进行冗余代码消除。

作业形式：　小组作业，要求两人一组合作完成

大作业问题描述

基础部分：词法分析与语法分析（14 分）

一、词法分析（7 分）：对关键字和变量名进行分析。要求:

1. 能够识别“return”、“def”和“var”三个关键字
2. 按照如下要求识别变量名：
    - 变量名以字母开头
    - 变量名由字母、数字和下划线组成
    - 变量名中有数字时，数字应该位于变量名末尾

例如：有效的变量名可以是 a123, b_4, placeholder 等。

无效的变量名可以是 b23c, b23_等

大作业问题描述

基础部分：词法分析与语法分析（14 分）

二、语法分析（7 分）：

对 Tiny 语言中一维或二维 Tensor 的声明及定义进行语法分析。语法必须符合以下要求：

1. 必须以“var”开头，后面为变量名及变量类型，最后为变量的初始化

2. 语法分析器已支持以下两种声明及初始化方式，以一个二维 Tensor 为例：

- var a = [[1, 2, 3], [4, 5, 6]];
- var a<2,3> = [1, 2, 3, 4, 5, 6];

需要同学们拓展 parser 功能支持第三种形式：

- var a[2][3] = [1, 2, 3, 4, 5, 6];

大作业问题描述

进阶部分：代码优化（6 分）

1. 代码优化：

进行冗余代码的消除。Tiny 语言内置的 transpose 函数会对矩阵进行转置操作。然而，对同一个矩阵进行两次转置运算会得到原本的矩阵，相当于没有转置。

2. 优化原因：

矩阵的转置运算是通过嵌套 for循环实现的，而嵌套循环是影响程序运行速度的重要因素。因此，侦测到这种冗余代码并进行消除是十分必要的。

```
def transpose_transpose(x) {
  return transpose(transpose(x));
}
```

目录 Contents

1 MLIR 编译框架简介

2 Tiny 语言简介

3 大作业问题描述

4 实验流程及演示

- ✓ 实验环境
- ✓ 构建步骤
- ✓ 基础部分的实现与验证
- ✓ 进阶部分的实现与验证
- ✓ 实验演示

实验环境

- 推荐操作系统：Ubuntu 20.04
- 推荐虚拟机软件：VMware workstation pro

构建步骤

- 依赖工具链：git, cmake, clang, lld, ninja
- sudo apt install git
- sudo apt install cmake
- sudo apt-get install clang lld
- sudo apt install ninja-build

构建步骤

- 安装MLIR：

- git clone https://github.com/Jason048/llvm-project.git
- mkdir llvm-project/build
- cd llvm-project/build
- cmake -G Ninja ../llvm \
  -DLLVM_ENABLE_PROJECTS=mlir \
  -DLLVM_BUILD_EXAMPLES=ON \
  -DLLVM_TARGETS_TO_BUILD="X86;NVPTX;AMDGPU" \
  -DCMAKE_BUILD_TYPE=Release \
  -DLLVM_ENABLE_ASSERTIONS=ON \
  -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ -DLLVM_ENABLE_LLD=ON

```
-- Targeting NVPTX
-- Targeting AMDGPU
-- Registering Bye as a pass plugin (static build: OFF)
-- Failed to find LLVM FileCheck
-- git version: v0.0.0 normalized to 0.0.0
-- Version: 1.6.0
-- Performing Test HAVE_THREAD_SAFETY_ATTRIBUTES -- failed to compile
-- Performing Test HAVE_GNU_POSIX_REGEX -- failed to compile
-- Performing Test HAVE_POSIX_REGEX -- success
-- Performing Test HAVE_STEADY_CLOCK -- success
-- Configuring done
-- Generating done
-- Build files have been written to: /share/llvm-project/build
```

构建步骤

编译MLIR：

- cmake --build . --target check-mlir

如果运行结果如下图所示，证明MLIR已经安装并成功编译

```
[0/1] Running the MLIR regression tests

Testing Time: 2.98s
  Unsupported:  223
  Passed     : 1116
```

安装tiny_project：

- git clone https://github.com/Jason048/tiny_project.git
- mkdir build
- cd build

编译tiny

- LLVM_DIR=/Path/to/llvm-project/build/lib/cmake/llvm \
  MLIR_DIR=/Path/to/llvm-project/build/lib/cmake/mlir \
  cmake -G Ninja ..

```
-- Generating done
-- Build files have been written to: /share/tiny_project/build
```

基础部分的实现与验证

词法分析器(/tiny_project/tiny/include/tiny/Lexer.h)：

在Lexer.h搜索"TODO"，可以看到需要补充的代码位置。 实现以下功能

1). 能够识别“return”、“def”和“var”三个关键字

2). 按照如下要求识别变量名：

- 变量名以字母开头
- 变量名由字母、数字和下划线组成
- 变量名中有数字时，数字应该位于变量名末尾

例如：有效的变量名可以是 a123, b_4, placeholder 等。

语法分析器(/tiny_project/tiny/include/tiny/Parser.h)：

在Parser.h搜索"TODO"，可以看到需要补充的代码位置。 实现以下功能

1). 语法变量必须以“var”开头，后面为变量名及类型，最后为变量的初始化

2). 语法分析器已支持以下两种初始化形式，以一个二维矩阵为例：

- var a = [[1, 2, 3], [4, 5, 6]]
- var a<2,3> = [1, 2, 3, 4, 5, 6]

需要同学们额外支持第三种新的形式：

- var a[2][3] = [1, 2, 3, 4, 5, 6]

□基础部分的实现与验证

词法分析器验证（test_1 - test_4）：验证当词法单元命名不规范时是否提示错误

- cmake --build . --target tiny
- cd ../
- build/bin/tiny test/tiny/parser/test_1.tiny -emit=ast

语法分析器验证（test_5 - test_6）：

test_5:验证语法分析器能否识别出三种声明及初始化方式，输出AST（-emit=ast）

- build/bin/tiny test/tiny/parser/test_5.tiny -emit=ast

test_6:执行以下指令查看输入程序的运行结果（-emit=jit）

- build/bin/tiny test/tiny/parser/test_6.tiny -emit=jit

□进阶部分的实现与验证

实现以下功能： 将tiny dialect的冗余转置代码优化pass补充完整。最终实现冗余代码的消除。

- build/bin/tiny test/tiny/parser/test_7.tiny -emit=mlir -opt

```
def transpose_transpose(x) {
  return transpose(transpose(x));
}

def main() {
  var a<2, 3> = [[1, 2, 3], [4, 5, 6]];
  var b = transpose_transpose(a);
  print(b);
}
```

□ 实验演示

谢谢！！