代码优化与生成实验报告

胡基宸 521021910143

在本次实验的过程中，我们完成了矩阵转置的优化，当使用嵌套的两次转置的时候，会得到原本的矩阵，相当于没有转置。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。

首先，我们用 getOperand 函数得到输入，用类型 Value 存储表示。

```
// TODO: Optimize the scenario: transpose(transpose(x)) -> x
// Step 1: Get the input of the current transpose.
// Hint: For op, there is a function: op.getOperand(), it returns the parameter of a TransposeOp and its type is mlir::Value.

/*
 *
 *  Write your code here.
 *
 */
Value input = op.getOperand ();
```

然后，用 getDefiningOp 函数比较输入的 input 是否是我们需要优化的嵌套转置，如果是，就进行后续优化，如果不是，返回 failure()

```
// Step 2: Check whether the input is defined by another transpose. If not defined, return failure().
// Hint: For mlir::Value type, there is a function you may use:
//       template<typename OpTy> OpTy getDefiningOp () const
//       If this value is the result of an operation of type OpTy, return the operation that defines it

/*
 *
 *  Write your code here.
 *  if () return failure();
 *
 */

auto inputOp = input.getDefiningOp<TransposeOp>();
if (!inputOp) return failure();
```

最后，我们用 replaceOp 函数将所有的 op，也就是检测到嵌套转置的地方，都转换成了 inputOp，也就是原本的矩阵，并返回 success()，表示优化成功。

```
// step 3: Otherwise, we have a redundant transpose. Use the rewriter to remove redundancy.
// Hint: For mlir::PatternRewriter, there is a function you may use to remove redundancy:
//       void replaceOp (mlir::Operation *op, mlir::ValueRange newValues)
//       The first argument will be replaced by the second argument.

/*
 *
 *  Write your code here.
 *
 */
rewriter.replaceOp (op , {inputOp.getOperand()});
return success();
```

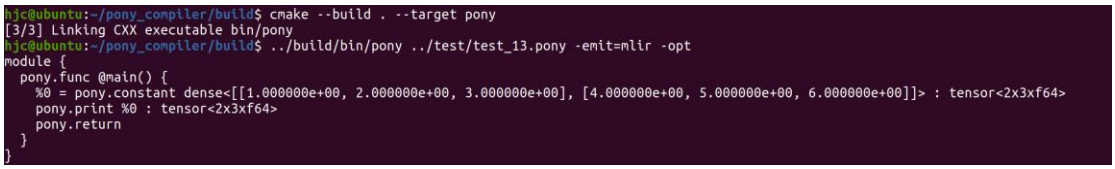
在完成上述代码优化功能后，运行测试用例 test_13。test_13 提供了一个冗余转置操作的实例，我们要求编译器能够通过代码优化去掉冗余的转置操作，输出优化后的结果。

对于 test_13 中的例子：

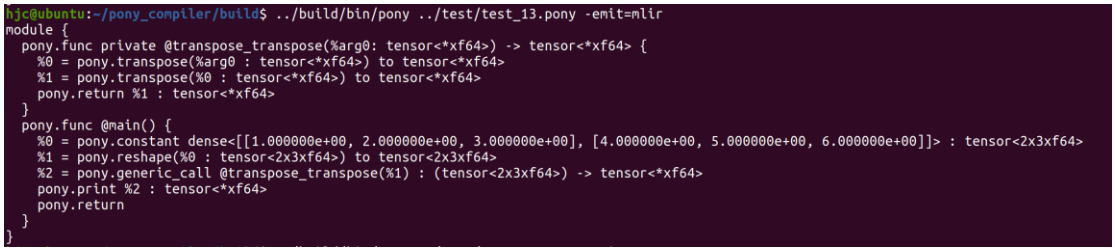
```
def transpose_transpose(x) {
  return transpose(transpose(x));
}

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

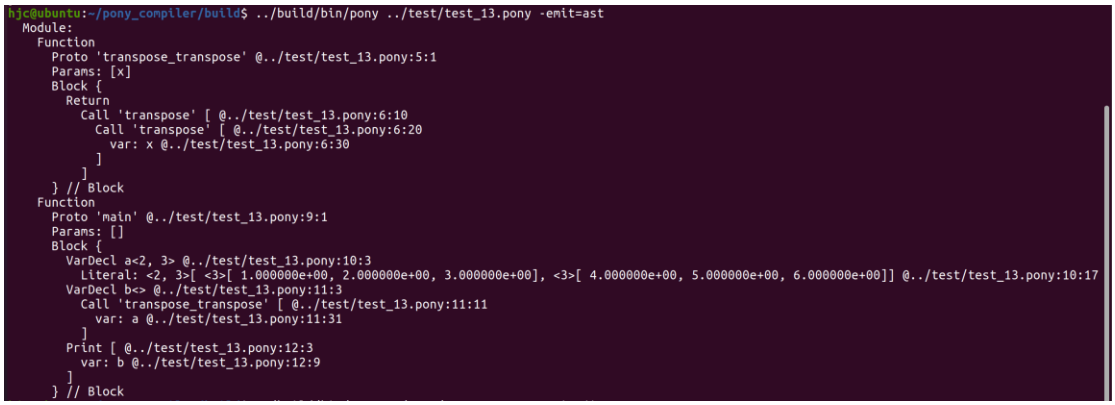
执行以下指令，输出转换后的 pony dialect（即转换后的代码表示）， 查看输出结果判断成功消除冗余转置。



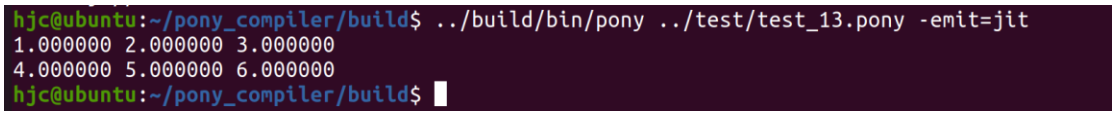
执行以下指令查看未优化的输出，对比优化前后输出的差异，发现少了两次不必要的转置。



还可以利用不同指令，输出此测试用例在编译过程中出现的多种形式，例如，将.pony 文件转换为抽象语法树 AST：



或直接得到 pony 程序的运行结果，是否进行消除冗余不会影响运行结果，发现运行结果与优化前一致。



本次代码量较少，主要是需要运用写好的 api 进行优化操作，对优化代码的操作有了更直观的理解。