

1. 实验过程

总共实现了4个pass，全部是FunctionPass：

1. Load和Store指令优化 class LoadStoreOpt

从根节点开始，采用DFS的方式遍历函数的支配树，当发现一条Store指令时：

1. 目的操作数不是Alloca指令，continue；
2. 目的操作数的Alloca类型为数组，continue；
3. 从当前指令Inst的下一条指令开始，采用DFS的方式遍历函数的支配树：
 1. 当前block与Inst所在的block不同并且当前block的前继大于1个（pred_size > 1），本节点return；
 2. 找到一条Store指令，并且目的操作数与Inst的目的操作数一致，本节点return；
 3. 找到一条Load指令，并且源操作数与Inst目的操作数一致，将Load指令之后所有的用法用Inst的源操作数替换（replaceAllUsesWith），然后将该Load指令删除。

2. 常量折叠 class ConstantFolding

从根节点开始，采用DFS的方式遍历函数的支配树，从而遍历函数的所有指令：

1. 指令Inst的opCode是二元操作符，lhs和rhs均为常数，则通过ConstantExpr::get(opCode, const1, const2)获得运算结果res，将该指令之后所有的用法用res替换，将Inst删除；
2. 指令的Inst的opCode是比较运算符FCmp或者ICmp，并且lhs和rhs均为常数，则通过ConstantExpr::getFCmp(predicate, const1, const2)或者ConstantExpr::getICmp(predicate, const1, const2)获得计算结果res，将该指令之后所有的用法用res替换，将Inst删除；

3. 死代码消除 class DeadCodeElimination

有两个部分：

1. 移除无用Alloca指令
遍历函数的entryBlock的所有Alloca指令，如果该指令的用法中没有Load，则将该指令的用法全部删除，然后将该指令删除
2. 移除其他无用指令
遍历函数的所有指令：
 1. 如果指令至少有一个用法，continue；
 2. 指令为RetInst/BrInst/CallInst/StoreInst，continue；
 3. 删除该指令

不断地运用1. 移除无用Alloca指令和2. 移除其他无用指令，直到函数不再变化。

4. 分支优化 class BranchOpt

有四个部分：

1. phi指令优化
遍历F的所有phi指令，如果phi指令中所有BasicBlock不是nullptr (<badref>) 的Value都是相同的，则将phi指令之后所有的用法都用该Value替换，该phi指令删除
2. 有条件跳转指令优化
遍历函数的所有有条件跳转指令（brInst.isConditional()）：
 1. 条件不为常量，continue；
 2. 将brInst改为无条件跳转指令
 1. 条件为真，跳转至brInst→getSuccessor(0)，如果brInst→getSuccessor(1)的第一条指令为phi指令，则phiInst-

- ```
>removeIncomingValue(brInst→getParent());
```
2. 条件为假, 跳转至`brInst→getSuccessor(1)`, 如果`brInst→getSuccessor(0)`的第一条指令为`phi`指令, 则`phiInst→removeIncomingValue(brInst→getParent());`
  3. 无用BasicBlock删除  
遍历函数的所有BasicBlock, 如果block的前继数量为0 (`pred_empty(block)`为真), 则将该block删除, 将block所有的后继`phi`指令用法设置为`nullptr`, `block→replaceSuccessorsPhiUsesWith(nullptr);`
  4. 无条件跳转指令优化  
遍历函数的所有无条件跳转指令(`brInst.isUnconditional()`), 如果跳转目标的前继数量为1, 则将跳转目标的所有指令全部复制到`brInst`所在的block, 将该`brInst`删除, 将跳转目标的后继`phi`指令用法设置为`nullptr`, 将跳转目标删除。

不断的运用1. `phi`指令优化、2. 有条件跳转指令优化、3. 无用BasicBlock删除、4. 无条件跳转指令优化, 直到函数F不再发生变化, 一般3.和4.后面要紧跟着1.

最终, pass的执行顺序为 `DeadCodeElimination → LoadStoreOpt → ConstantFolding -> LoadStoreOpt → ConstantFolding → DeadCodeElimination → BranchOpt`

## 2. 遇到的难点及解决方法

上述4个pass的实现逻辑都是挺简单的, 并且Load和Store指令优化采用的也是比较保守的方法, 我也没有系统地使用数据流分析。

本次实验遇到的最大的困难就是在BranchOpt中, 删除无用BasicBlock和删除跳转目标后`phi`指令出现`<badref>`, block被删除了, 但是`phi`指令中对应的block指针不是`nullptr`, 而是一个地址, 因此使用`getIncomingBlock(i) == nullptr`来判断`<badref>`不行, 后来在IDE中搜索`replace`时突然发现了`replaceSuccessorsPhiUsesWith()`这个函数, block被删除前调用该函数, 传入参数`nullptr`, 那么`phi`指令中所有的`<badref>`都是`nullptr`了。

## 3. 成绩

相比于实验3, 性能分从0.169提升到了0.259, 通过的算例从634增加到了638。

我发现, LoadStoreOpt和DeadCodeEliminationOpt结合起来就可以使性能分达到0.24多, 所以在本次实验中ConstantFolding和BranchOpt对代码性能的提升非常少。

### 测试报告

排行榜分数

performance 0.25939452440309374

score 638

提交排行

| Autograder <span>🏠</span> |                     |                     |       |
|---------------------------|---------------------|---------------------|-------|
| 作业详情 (#31)                |                     |                     |       |
| 提交记录                      | 排行榜                 |                     |       |
| 排行榜                       |                     |                     |       |
| ← 回到作业列表                  |                     |                     |       |
| 名次                        | 提交时间                | performance         | score |
| 11                        | 2023-06-14 16:29:04 | 0.29665272186288697 | 645   |
| 12                        | 2023-06-18 03:10:33 | 0.25999452440309374 | 638   |
| 13                        | 2023-06-09 15:16:29 | 0.2420917261194229  | 633   |
| 14                        | 2023-05-22 23:24:20 | 0.2327925944689375  | 639   |
| 15                        | 2023-05-26 12:54:58 | 0.22373143827305925 | 639   |
| 16                        | 2023-06-14 18:08:45 | 0.20941975959459533 | 637   |
| 17                        | 2023-05-16 19:06:03 | 0.20284044371482365 | 636   |
| 18                        | 2023-06-07 16:07:51 | 0.188908677723633   | 627   |
| 19                        | 2023-05-25 02:43:13 | 0.18338986175420388 | 636   |
| 20                        | 2023-06-06 18:11:25 | 0.17815198742387994 | 637   |

dead-code-elimination-3.sysu.c和integer-divide-optimization-2.sysu.c均在时限内通过:

...ance/performance\_test2021-private/dead-code-elimination-3.sysu.c (1/1)

```
Compile Finish.
[478/647] clang -O3: 18712us, ret 0
[478/647] sysu-lang: 286217us, ret 0
```

...erformance\_test2021-private/integer-divide-optimization-2.sysu.c (1/1)

```
Compile Finish.
[492/647] clang -O3: 85223us, ret 0
[492/647] sysu-lang: 32301896us, ret 0
```

而integer-divide-optimization-3.sysu.c和hoist-3.sysu.c均未在时限内通过:

...erformance\_test2021-private/integer-divide-optimization-3.sysu.c (0/1)

```
Compile Finish.
[493/647] clang -O3: 1363454us, ret 0
TimeoutExpired.
```

...g-tester-performance/performance\_test2021-private/hoist-3.sysu.c (0/1)

```
Compile Finish.
[487/647] clang -O3: 4724200us, ret 0
TimeoutExpired.
```