Project 6 - Optimization (Optional)

2025 年 12 月 16 日

1 项目要求

在 Project 6 中，你需要对你的编译器进行优化，实现 SSA 与 Mem2Reg 优化。

1.1 代码要求与评分

在本次 Project 中，教学团队**不会准备也不会提供任何测试样例、framework 代码、自动测评服务**，你可以随意修改代码（包括 `framework` 包）；你需要自己撰写测试样例来证明你的编译器实现了上述功能，并且能够产生合法有效的 LLVM IR。

本次 Project 满分为 50 分，仅用于弥补你在之前 Project 丢失的分数，并不能溢出至总评。

1.2 评分方式

本次 Project 没有评测，你需要向助教当面展示你的 Project 并证明它按预期工作。请在 2026 年 1 月 5 日 20:00 前向助教发送邮件预约时间，展示时间不晚于 2026 年 1 月 9 日晚 18:00。

邮件请发送至 `chenjf@mail.sustech.edu.cn`，并 CC 你们所有的小组成员。

2 Mem2Reg 介绍

> `https://www.llvm.org/docs/Passes.html#mem2reg-promote-memory-to-register`
> This file promotes memory references to be register references. It promotes alloca instructions which only have loads and stores as uses. An alloca is transformed by using dominator frontiers to place phi nodes, then traversing the function in depth-first order to rewrite loads and stores as appropriate. This is just the standard SSA construction algorithm to construct “pruned” SSA form.

在 Project 5 中，对于开辟在栈上的局部变量，我们对它的处理方式是用 `alloca` 分配内存空间，然后使用 `load` 与 `store` 指令进行读取与写入。在翻译 LLVM IR 到汇编的时候，这些指令会分别被翻译为 `sub rsp, xxx` 与基于栈指针（`rsp` 或 `rbp`）寻址的 `mov` 访存指令。但有的时候，对这些局部变量的读写可以仅在寄存器（Register）中完成，不需要将结果放置在栈上。

例如，在以下代码中，按照我们 Project 5 的逻辑，`result` 会对应一个由 `alloca` 指令分配的栈空间，然后在两个分支中使用 `store` 修改内存空间中的值。

```
int max(int a, int b) {
    int result;
    if (a > b) {
        result = a;
    } else {
        result = b;
    }
    return result;
}
```

而经过 Mem2Reg 优化后的 LLVM IR 可以删除掉 `result` 的 `alloca` 指令；在 SSA 形式下，我们无法对 `%result.0` 进行多次赋值，取而代之的方法是使用 phi 节点中根据来源分支选择 `result` 的值。

```
define i32 @max(i32 %a, i32 %b) {
entry:
  ; result 的 alloca 被删除了
  %cmp = icmp sgt i32 %a, %b
  br i1 %cmp, label %if.then, label %if.else

if.then:
  ; store 被删除了，值就在 %a 里
  br label %if.end

if.else:
  ; store 被删除了，值就在 %b 里
  br label %if.end

if.end:
  ; 插入 PHI 节点：如果控制流来自 if.then，取 %a；如果来自 if.else，取 %b
  %result.0 = phi i32 [ %a, %if.then ], [ %b, %if.else ]

  ret i32 %result.0
}
```

2.1 使用 opt 观察 mem2reg

你可以使用 LLVM 套件中的 `opt` 工具单独执行 `mem2reg` 这个优化 Pass：

```
opt -S -passes=mem2reg source.ll -o output.ll
```

如果你的 `source.ll` 是使用 `clang` 加上 `-O0` 参数生成的，请注意给 `clang` 加上一个新的参数`-Xclang -disable-O0-optnone`。否则，`clang` 生成的 ll 文件中会存在 `attributes #0 = { ... optnone ... }` 这样一个 attributes，这会阻止 `opt` 对函数的优化。

3 测试要求

在你的展示中，你需要：

- 证明你的 mem2reg 优化可以在 if 条件分支、while 循环中工作。
- 证明你的编译器在启用 mem2reg 的情况下仍然能通过 Project 5 中的 Official Tests。
- 解释什么时候不应该对 `alloca` 进行 `mem2reg` 优化。