基于LLVM的SysYF进阶优化

李牧龙 徐航宇

中国科学技术大学
University of Science and Technology of China

实验内容

- 将22年PW8框架合并入PW6中
- 采用多种方式进行LLVM中间代码的优化
- 后端生成ARM代码

代码框架合并

代码框架合并

- 裸指针修改为智能指针

```
void Mem2Reg::insideBlockForwarding(){
    for(auto bb: func_->get_basic_blocks()){
        std::map<Value *, Instruction *> defined_list;
        std::map<Instruction *, Value *> forward_list;
        std::map<Value *, Value *> new_value;
        std::set<Instruction *> delete_list;
        for(auto inst: bb->get_instructions()){
            if(!isLocalVarOp(inst))continue;
            if(inst->get_instr_type() == Instruction::OpID::store){
                Value* lvalue = static_cast<StoreInst *>(inst)->get_lval();
                Value* rvalue = static_cast<StoreInst *>(inst)->get_rval();
                auto load_inst = dynamic_cast<Instruction*>(rvalue);
                if(load_inst && forward_list.find(load_inst) != forward_list.end()){
                    rvalue = forward_list.find(load_inst)->second;
                }
                if(defined_list.find(lvalue) != defined_list.end()){
                    auto pair = defined_list.find(lvalue);
                    delete_list.insert(pair->second);
                    pair->second = inst;
                }
                else{
                    defined_list.insert({lvalue, inst});
                }
                if(new_value.find(lvalue) != new_value.end()){
                    new_value.find(lvalue)->second = rvalue;
                }
                else{
                    new_value.insert({lvalue, rvalue});
                }
            }
```

```
void Mem2Reg::insideBlockForwarding(){
    for(auto bb: func_->get_basic_blocks()){
        std::map<Ptr<Value>, Ptr<Instruction>> defined_list;
        std::map<Ptr<Instruction>, Ptr<Value>> forward_list;
        std::map<Ptr<Value>, Ptr<Value>> new_value;
        std::set<Ptr<Instruction>> delete_list;
        for(auto inst: bb->get_instructions()){
            if(!isLocalVarOp(inst))continue;
            if(inst->get_instr_type() == Instruction::OpID::store){
                Ptr<Value> lvalue = static_pointer_cast<StoreInst>(inst)->get_lval();
                Ptr<Value> rvalue = static_pointer_cast<StoreInst>(inst)->get_rval();
                auto load_inst = dynamic_pointer_cast<Instruction>(rvalue);
                if(load_inst && forward_list.find(load_inst) != forward_list.end()){
                    // 该指令是load指令，且其值已经被替换
                    rvalue = forward_list.find(load_inst)->second;
                }
                if(defined_list.find(lvalue) != defined_list.end()){
                    // 该变量已经被定义过，需要删除上一条定义
                    auto pair = defined_list.find(lvalue);
                    delete_list.insert(pair->second);
                    pair->second = inst;
                }
                else{
                    // 该变量未被定义过，直接插入
                    defined_list.insert({lvalue, inst});
                }
                if(new_value.find(lvalue) != new_value.end()){
                    new_value.find(lvalue)->second = rvalue;
                }
                else{
                    new_value.insert({lvalue, rvalue});
                }
            }
```

中间代码优化

主要内容

- Mem2Reg
- **常量折叠与传播**
- **函数性质分析**
- **局部公共子表达式消除**
- **死代码删除**
- **循环表达式外提**
- **活跃变量分析**

Mem2Reg

- 代码框架中提供的优化，将局部非数组变量放入寄存器中，消除相关访存
- 在基本块内部，对于每一个load，用对应变量的最近一次store替换其使用
- 在基本块间，需要在不同定值汇合处增加phi指令
- 需要计算支配树和支配边界

常量折叠与传播

- Mem2Reg优化产生大量的常数表达式
- 在编译时计算所有的常量表达式
- 可以减少条件分支

函数性质分析

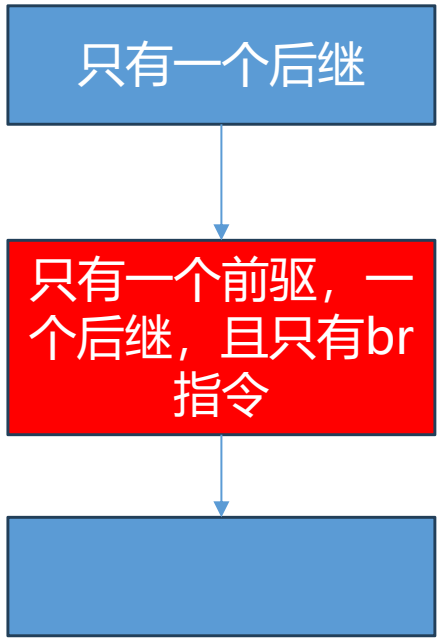
- 分析是否为纯函数
  - 纯函数即返回值仅依赖于参数，且不产生副作用的函数
- 简单起见，采用下面这个定义：
  - 没有引用全局变量/对全局变量赋值
  - 没有传入数组参数
  - 调用的函数全为纯函数
  - 外部函数视为非纯函数

局部公共子表达式消除

- 合并同一基本块内相同表达式
- 不考虑alloca, store, load
- 比较运算类型、操作数是否相同即可

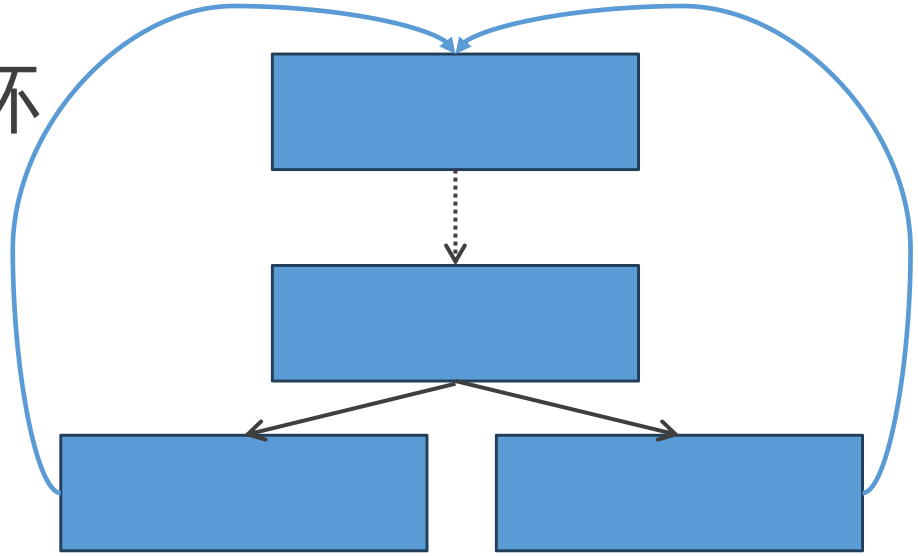
死代码删除

- 分为死指令删除和死基本块删除
- 死指令删除：
  - 删除未被引用的指令
  - 检查use链是否为空即可
- 死基本块删除：
  - 删除从entry开始dfs，未经过的基本块
  - 删除形如右图（直上直下，只有一条br指令）的基本块：
  - 需要维护phi指令



循环表达式外提

- 通过回边寻找自然循环
- 特殊情况：



- 将未引用循环内定值的指令提至循环外
- 不考虑alloca, br, store, load, phi
- 可能引入新基本块

活跃变量分析

- 后端寄存器分配的基础
- 大体上按课内所讲的数据流进行分析
  - **活跃变量数据流等式**
    - $\mathbf{IN}\ [B] = use_B \cup (\mathbf{OUT}\ [B] - def_B)$
    - $\mathbf{OUT}[B] = \cup_{S\text{是}B\text{的后继}}\ \mathbf{IN}\ [S]$
    - $\mathbf{IN}\ [\mathbf{EXIT}] = \varnothing$
- 由于phi指令，增加特别处理：维护活跃变量来源
- 如%op3 = phi [%op1, label1], [%op2, label2]，那么%op1只在label1出口活跃

效果展示

- 常量折叠与传播+死代码删除

```
define i32 @if_ifElse_() {
label_entry:
  %op4 = icmp eq i32 5, 5
  br i1 %op4, label %label6, label %label9
label_ret:                                        ; pred
  ret i32 %op16
label6:                                           ; preds =
  %op8 = icmp eq i32 10, 10
  br i1 %op8, label %label11, label %label12
label9:                                           ; preds =
  %op16 = phi i32 [ 5, %label_entry ], [ %op17, %label15 ]
  br label %label_ret
label11:                                          ; preds
  br label %label15
label12:                                          ; preds
  %op14 = add i32 5, 15
  br label %label15
label15:                                          ; preds
  %op17 = phi i32 [ %op14, %label12 ], [ 25, %label11 ]
  br label %label9
}
define i32 @main() {
label_entry:
  %op1 = call i32 @if_ifElse_()
  br label %label_ret
label_ret:                                        ; pred
  ret i32 %op1
}
```

```
define i32 @if_ifElse_() {
label_entry:
  br label %label_ret
label_ret:
  ret i32 25
}
define i32 @main() {
label_entry:
  %op1 = call i32 @if_ifElse_()
  br label %label_ret
label_ret:
  ret i32 %op1
}
```

效果展示

- 循环表达式外提

```
define i32 @add(i32 %arg0, i32 %arg1) {
label_entry:
  %op7 = add i32 %arg0, %arg1
  br label %label_ret
label_ret:                                        ; pr
  ret i32 %op7
}
define i32 @main() {
label_entry:
  br label %label6
label_ret:                                        ; pr
  ret i32 %op22
label6:                                          ; preds
  %op22 = phi i32 [ 0, %label_entry ], [ %op25, %label19 ]
  %op23 = phi i32 [ 0, %label_entry ], [ %op26, %label19 ]
  %op24 = phi i32 [ 0, %label_entry ], [ %op21, %label19 ]
  %op8 = icmp slt i32 %op24, 10
  br i1 %op8, label %label9, label %label10
label9:                                          ; preds
  br label %label12
label10:                                          ; pred
  br label %label_ret
label12:                                          ; pred
  %op25 = phi i32 [ %op22, %label9 ], [ %op16, %label15 ]
  %op26 = phi i32 [ 0, %label9 ], [ %op18, %label15 ]
  %op14 = icmp slt i32 %op26, 10
  br i1 %op14, label %label15, label %label19
label15:                                          ; pred
  %op16 = call i32 @add(i32 1, i32 2)
  %op18 = add i32 %op26, 1
  br label %label12
label19:                                          ; pred
  %op21 = add i32 %op24, 1
  br label %label6
}
```

```
define i32 @add(i32 %arg0, i32 %arg1) {
label_entry:
  %op7 = add i32 %arg0, %arg1
  br label %label_ret
label_ret:                                        ;
  ret i32 %op7
}
define i32 @main() {
label_entry:
  %op16 = call i32 @add(i32 1, i32 2)
  br label %label6
label_ret:                                        ;
  ret i32 %op22
label6:                                          ; pr
  %op22 = phi i32 [ 0, %label_entry ], [ %op25, %label19 ]
  %op24 = phi i32 [ 0, %label_entry ], [ %op21, %label19 ]
  %op8 = icmp slt i32 %op24, 10
  br i1 %op8, label %label9, label %label10
label9:                                          ; pr
  br label %label12
label10:                                          ; p
  br label %label_ret
label12:                                          ; p
  %op25 = phi i32 [ %op22, %label9 ], [ %op16, %label15 ]
  %op26 = phi i32 [ 0, %label9 ], [ %op18, %label15 ]
  %op14 = icmp slt i32 %op26, 10
  br i1 %op14, label %label15, label %label19
label15:                                          ; p
  %op18 = add i32 %op26, 1
  br label %label12
label19:                                          ; p
  %op21 = add i32 %op24, 1
  br label %label6
}
```

效果展示

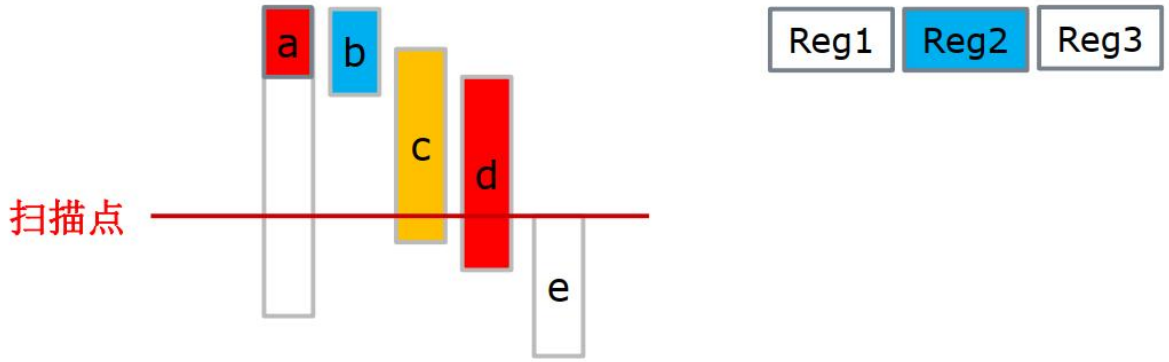
- 局部公共子表达式消除

```
define i32 @add(i32 %arg0, i32 %arg1) {
label_entry:
  %op7 = add i32 %arg0, %arg1
  br label %label_ret
label_ret:
  ret i32 %op7
}
define i32 @main() {
label_entry:
  %op5 = call i32 @add(i32 1, i32 2)
  %op9 = call i32 @add(i32 1, i32 2)
  %op13 = call i32 @add(i32 %op5, i32 %op9)
  br label %label_ret
label_ret:
  ret i32 %op13
}
```

```
define i32 @add(i32 %arg0, i32 %arg1) {
label_entry:
  %op7 = add i32 %arg0, %arg1
  br label %label_ret
label_ret:
  ret i32 %op7
}
define i32 @main() {
label_entry:
  %op5 = call i32 @add(i32 1, i32 2)
  %op13 = call i32 @add(i32 %op5, i32 %op5)
  br label %label_ret
label_ret:
  ret i32 %op13
}
```

后端代码生成

寄存器分配



- 线性扫描寄存器分配
- 按照活跃区间的左端点进行升序排列
- 依次尝试分配寄存器：
  - 直接分配空闲寄存器
  - 分配已退出活跃区间的寄存器
  - 与其他变量争夺寄存器（标准：早结束者获胜）

phi指令数据搬移

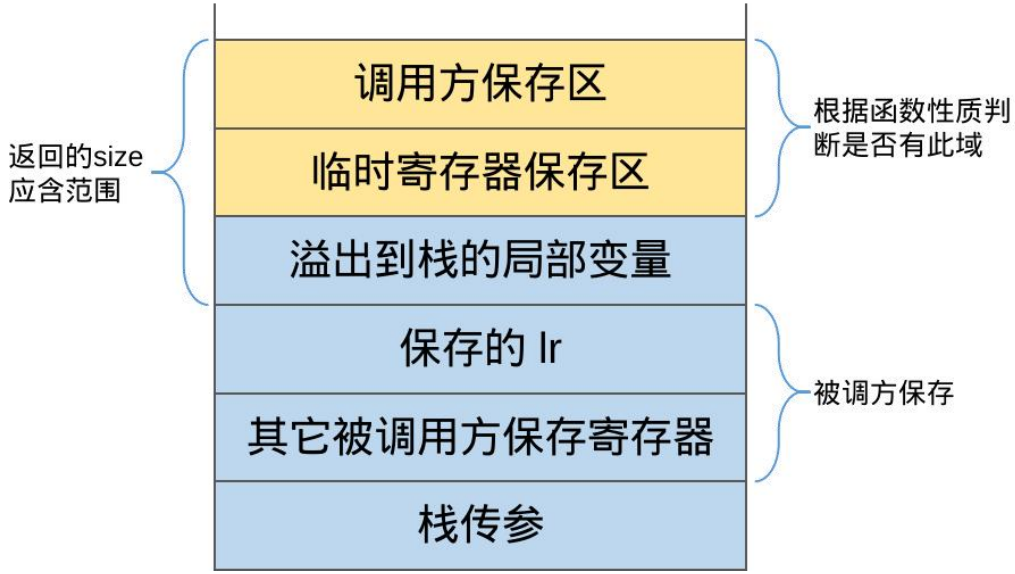
- phi指令要求对若干组变量进行并行的数据搬移
- 建立依赖图模型（src->dst)
- 观察：所有节点入度至多为一

  ==> 依次删除出度为0的节点后，剩余图为若干圈的并

phi指令数据搬移

- 固定临时寄存器r10,r12
- 搬移过程：
  - 逆拓扑排序，找到出度为0的结点，直接搬移
  - 从图中剩余的任意点出发做DFS，得到圈
  - 按照圈的顺序依次搬移数据

栈分配



- 按照ARM的ABI约定组织栈空间
- 为溢出的局部变量、参数、数组分配空间
- 维护stack_map，并计算sp的跳转大小

浮点数适配

- 添加浮点运算指令、浮点比较指令、类型转换指令
- 为浮点指令添加合适的后端代码生成
- 遇到的问题：
  - ARMv8的浮点指令对使用的寄存器类型有严格要求

测试结果

中间代码优化

- 使用clang作为后端
- 正确性：正确通过了PW8框架中所有测试样例
- 运行时间：所选样例(Hard_H)上，运行时间缩短约11.5%

后端代码生成

- 在树莓派(ARMv8)上使用clang编译汇编代码
- 编写python脚本进行批量测试
- 正确性：
  - 截至1月21日22点，PW8框架中测试样例的通过率为133/140
  - 后经修改通过了所有测试样例