Decaf PA 5

王世因 2016011246

实验的主要代码

我沿用了在PA4中给`Tac`增加的`def`属性，如果这条语句定义了变量，那么`def`就存储那个变量的值，否则就是`null`

- backend/GraphColorRegisterAllocator.java

```
for(Temp liveuse: bb.liveUse){
    load(bb.tacList, liveuse);
}
this.inferenceGraph.alloc(this.bb, regs, this.fp.reg);
this.inferenceGraph.makeGraph();
```

- backend/InferenceGraph.java

```
for(Temp t:bb.liveUse){
   addNode(t);
}

for(Temp t1:bb.liveUse){
   for(Temp t2:bb.liveUse){
      if(t1!=t2){
         addEdge(t1, t2);
      }
   }
}

for (Tac tac = bb.tacList; tac != null; tac = tac.next) {
   if(tac.def!=null && tac.liveOut!=null){
      addNode(tac.def);
      for(Temp t:tac.liveOut){
         addNode(t);
         addEdge(tac.def, t);
      }
   }
   switch (tac.opc) {......}
}
```

基本块儿中两个节点连边的条件

1. 所有在基本块儿`LiveUse`集合中的变量都要连边，构成完全图

2. 对于每一条语句，如果它`Def`定义/更新了某一个变量，那么就要把这个变量和本条语句对应的所有的`LiveOut`中的点相连

完整的干涉图染色的寄存器分配算法

算法来源："Register allocation via coloring"

图的最少染色问题是NP-hard的，现有的解决方法是使用Heuristic来求解。每次选择小于现有寄存器个数的点暂时删除，归为一种颜色，然后进行递归直到最后只剩一个点。再往回逐个染色。

```
While(the graph has more than one node){
    Pick a node t with fewer than k neighbors
    Put t on a stack and remove it from the RIG
}
```

有时寄存器不够用，我们就要利用内存空间来存储，每次都要把变量`store`或者`load`到寄存器。选择哪个点做这个操作Spilling就需要通过Heuristic的方法来判断。这部分的优化应该添加到Inference中，同时我们也需要在Allocation的类中完善下面的代码，在`temp.reg != null`时，也可能是干涉图没有找到合适的寄存器的情况。

```
private void findReg(Tac tac, Temp temp, boolean read) {
    // We've done register allocation before, so here we bind register and
values.
    if (temp.reg != null) {
        temp.reg.var = temp;
        return;
    }
    throw new IllegalArgumentException("Register allocation incomplete!");
}
```

同时在InferenceGraph中修改这部分的代码：

```
private boolean color() {
    if (nodes.isEmpty())
        return true;

    // Try to find a node with less than K neighbours
    Temp n = null;
    for (Temp t : nodes) {
        if (nodeDeg.get(t) < regs.length) {
            n = t;
            break;
        }
    }

    if (n != null) {
        // We've found such a node.
        removeNode(n);
```

```
            boolean subColor = color();
            n.reg = chooseAvailableRegister(n);
            return subColor;
        } else {
            throw new IllegalArgumentException(
                "Coloring with spilling is not yet supported");
        }
    }


    Register chooseAvailableRegister(Temp n) {
        Set<Register> usedRegs = new HashSet<>();
        for (Temp m : neighbours.get(n)) {
            if (m.reg == null) continue;
            usedRegs.add(m.reg);
        }
        for (Register r : regs)
            if (!usedRegs.contains(r))
                return r;
        return null;
    }
```

遇到的问题和实验体会

通过第五次书面作业，我熟悉了干涉图寄存器分配的基本原理，理解了BasicBlock和FlowGraph的数据结构，对写代码起到了很重要的辅助作用。可见，在写代码前，需要提前熟悉实验的原理，否则事倍功半。

本次试验的报错含有的信息量比较少，不能像之前的PA一样直接通过输出的报错来debug，虽然代码量不大但也花了一些时间debug。一开始我使用错了函数，用了`GraphColorRegisterAllocator.bind`，后来才通过分析框架以及和同学讨论的方式换成了`GraphColorRegisterAllocator.load`。

至此编译原理的PA都完成了，感谢老师和助教这一学期以来的帮助，我通过咱们的课堂和作业收获良多！