Decaf PA5 Report

王琛 计65 2016011360

一、本阶段工作

本阶段的修改都在`InferenceGraph.java`和`GraphColorRegisterAllocator`中，代码量并不大。

1. 求解思路

这次的实验是要求通过对干涉图染色的方法来实现寄存器的分配。从数据流分析的结果构建基本块的干涉图，然后利用一个启发式算法对干涉图染色，最终根据染色结果分配寄存器。其中，需要完成的是构建基本块干涉图的部分。干涉图的节点表示的是一个Temp变量，连边的条件是两个节点不能同时使用一个寄存器。而两个变量何时不能使用一个寄存器？当一个变量被定值时，它的值被放入寄存器中。而在这条定值语句之后仍然活跃的变量，其值也应该在寄存器中。这时，被定值变量和其后活跃的变量就应该连一条边。另外，在基本块开始之前，这个块的liveUse集合需要被装入寄存器中，这些变量之间也应该连边。即连边条件概括为：

- 定值语句被定值变量和这条TAC的liveOut中变量
- 此基本块的liveUse集合中的变量两两相连

2. 具体实现

`InferenceGraph.java`：makeEdges函数根据上述加边条件进行加边。伪码描述为：

```
for all tac in the basicblock which defines a variable:
    for all variable v in the liveOut of this tac:
        if v is a node in this bb:
            addEdge(tac.op0, v);
        end
    end
end
for variable in the liveUse of this basicblock:
    addEdge mutually
end
```

`GraphColorRegisterAllocator.java`：alloc函数中调用`InferenceGraph`进行寄存器分配，并将liveUse集合加载到寄存器。加载liveUse到寄存器应该在分配完寄存器之后进行。伪码描述为：

```
call alloc in InferenceGraph
load variable to regs for all in liveUse of basicblock
```

二、完整的干涉图染色寄存器分配

实现思路：根据龙书的描述，"an edge connects two nodes if one is live at a point where the other is defined"。因此还是扫描TAC语句，如果这是一条定值语句，而其后还有活跃变量，那么被定值变量和后面的活跃变量要连一条边。因为是按函数分配，就不用考虑基本块的liveUse问题。另外，还需要考虑是否泄漏到内存的问题，需要为Temp变量加入相应的标记。因此，在染色时，如果找不到度数小于K的，将其泄漏到内存中。

伪码描述：

`InferenceGraph.java`：

```
makeNodes:
for all tac in the procedure:
    addNode for each new variable
end

makeEdges:
for all tac in the procedure that defines a variable:
    for all variable v in the liveOut of this tac:
        addEdge(tac.op0, v);
    end
end

alloc:
if node is empty:
    return;
end
find a node n whose degree is less than K:
    remove n and its edges
    call alloc
    n.reg = chooseAvailableRegister(n)
    return;
end
choose the node n that has largest degree:
    remove n and its edges
    call alloc
    mark n is spilled into memory
end
```

`GraphColorRegisterAllocator.java`：

```
alloc:
init an InferenceGraph object
call alloc of InferenceGraph
```

`Mips.java`：更换原来的genAsmForBB函数，如果遇到被定值的变量在内存中，先找到一个暂时不用的寄存器存放该变量，完成后存入内存再恢复；如果遇到被使用的变量在内存中，同样找一个这条语句不用的寄存器，从内存中读入，完成后恢复。

```
emitAsm:
for all procedure:
    alloc regs
    call genAsmGlobally

genAsmGlobally:
for all tac in the procedure:
    if Temp t in tac is spilled into memory:
        if t is to be defined:
            find available reg R0
            push R0
            save t to R0
            store R0
            pop R0
        end
        if t is referenced:
            find available reg R0
            push R0
            restore t from memory to R0
            pop R0 when finished
        end
    end
end
```

Acknowledgement

这次实验中，李文凯同学、李映辉同学给予了我帮助，在此表示感谢！