PA5实验报告

2017011307 张晨

算法流程

1. 建立干涉图，其细节见“问题回答”篇
2. 对干涉图进行染色。染色算法为从小到大枚举k，不断迭代删除图中度数小于k的点，若能删成空图，则说明可以k染色。之后按删除顺序的逆序为节点添加颜色，保证已添加的颜色互相不冲突。
3. 按染色结果进行最终汇编代码的生成。在函数开始时，从寄存器直接拷贝所有参数（因为框架本身不支持超过4个参数，我也就没有考虑用栈传递参数的情况）。之后顺次添加各个指令，将Temp替换为实际寄存器。因为寄存器是够用的，所以仅在call的时候需要将寄存器值暂存到栈里，其他时候染色算法可以保证寄存器没有冲突。

问题回答

- 如何确定干涉图的节点?

  干涉图的节点包括函数的所有参数，每个块的`LiveIn`，`def`，并进行去重。

- 连边的条件是什么?

  函数的所有参数之间，每个块的所有`LiveIn`之间，每个指令的目标寄存器和它的`LiveOut`之间。

结果分析

测例`basic-math`中的`pow`函数。

```
static int pow(int a, int b) {
    int result = 1;
    for (int i = 0; i < b; i = i + 1) {
        result = result * a;
    }
    return result;
}
```

贪心算法生成的汇编为

```
_L_Maths_pow:  # function FUNCTION<Maths.pow>
    addiu   $sp, $sp, -44
    sw      $a0, 0($sp)
    sw      $a1, 4($sp)
    li      $v1, 1
    move    $t0, $v1
    li      $v1, 0
    move    $t1, $v1
    sw      $t0, 36($sp)
    sw      $t1, 40($sp)
_L4:
```

```
    lw      $v1, 40($sp)
    lw      $t0, 4($sp)
    slt     $t1, $v1, $t0
    sw      $t0, 4($sp)
    sw      $v1, 40($sp)
    beqz    $t1, _L3
    lw      $v1, 36($sp)
    lw      $t0, 0($sp)
    mul     $t1, $v1, $t0
    move    $v1, $t1
    li      $t1, 1
    lw      $t2, 40($sp)
    add     $t3, $t2, $t1
    move    $t2, $t3
    sw      $t0, 0($sp)
    sw      $v1, 36($sp)
    sw      $t2, 40($sp)
    j       _L4
_L3:
    lw      $v1, 36($sp)
    move    $v0, $v1
    j       _L_Maths_pow_exit

_L_Maths_pow_exit:
    addiu   $sp, $sp, 44
    jr      $ra
```

着色算法生成的汇编为

```
_L_Maths_pow:
    addiu   $sp, $sp, -36
    move    $t2, $a0
    move    $t1, $a1
    li      $v1, 1
    move    $t0, $v1
    li      $v1, 0
    move    $v1, $v1
_L4:
    slt     $t3, $v1, $t1
    beqz    $t3, _L3
    mul     $t0, $t0, $t2
    move    $t0, $t0
    li      $t3, 1
    add     $v1, $v1, $t3
    move    $v1, $v1
    j       _L4
_L3:
    move    $v0, $t0
    j       _L_Maths_pow_exit
_L_Maths_pow_exit:
    addiu   $sp, $sp, 36
    jr      $ra

```

由于进行了着色，进出每个基本块的时候不再需要将寄存器从栈中读出/存到栈中，因此指令条数大大减少，且减少的还是较慢的访存指令。在这个例子中，两个算法实际使用的寄存器数量没有太大差异，原因是每个基本块都很简单。总的来说，着色算法的效果更好。