申优文章

16231137 李航宇

**写在前面：**

这篇申优文章这是介绍一下自己一路走来在编译课设里面的收获与成长，同时写一下自己做的优化部分以及对编译课设未来的展望。

**收获与成长：**

作为一个只经历过计组的高工学生来说，最开始对编译的印象也就是感觉和计组一样，每一周老师给需求，我们照着课程网站或是指导书进行操作同时验证工程可靠性与正确性，而且编译课设开始的时间比较晚，所以我最初的感觉就是编译这门课应该远没有计组那么恐怖，但是当自己真正做的时候才发觉编译课设可以说是一个真正意义上的课程“设计”！之前的东西无非都被限定在指导书与老师的条条框框中，有统一的测试集进行测试，给人一种你如果按部就班就可以做得好的感觉，但是编译这门课的需求就不像计组那样的细致，它有一个明确的大需求—简化的C0文法编译器设计，但是也仅仅就建立在这样一个包罗万象的需求上。没有指导书，没有明确测试集，没有绝对的正确与错误，这些都是因约定而异的。所以我才说这门课是最接近设计的一门课。就在这样一个啃硬骨头的课程上，我真正意义上感受到了按照具体规则实现算法与按照模糊需求实现代码之间巨大的差异性。可能小一点一两百行的程序还看不出差距，但是在编译课设这样5000+行的代码的基础上，二者之间的鸿沟就被无限放大了。我也理解了计组时高小鹏老师一直强调的架构到底有多重要，现在想一下自己如果能够在实现优化的时候再想得仔细一些或者再整合一下数据结构，可能最终结果会很漂亮吧。但是人生也没有重来，学到了这一点的我之后也就不会再走这个弯路了。

**我都做了哪些优化？**

实现了常量折叠，dag图公共子表达式删除，活跃变量分析，着色图法进行寄存器分配，以及小部分窥孔优化。

#### 常数合并

在语法分析时，程序会解析出表达式中各项的单词类型，此时的三个语法分析函数expr，item以及fator可以通过递归调用将常量进行合并。

在三个函数进行分析时分别传入\*is\_const指针，\*value指针与\*tmp\_name，两个指针分别能从递归的语法分析项获取底层常数分析结果，其中is\_count为真时，value中会传回底层计算获得的常数值，当is\_count为假时，底层会产生一个临时变量名，同时将底层处理的运算写入四元式vector中，向上返回临时变量记号。如果向上传递后某运算符左右的变量均可确定为常量，就用这个常量代替这个表达式，这样经过一层一层地向上传递就可以将常量折叠，但是缺点是这种阅读代码的方式是从左向右，从上至下的，所以当一个表达式第一个不是常量而后面的都是常量，比如a = b + 1 / 3等等，这种方法对后面可以合并的常量就无能为力了。

#### 基本块的划分

基本块的划分按照书上的算法的修改版来做，因为书上并不支持函数调用等中间代码的形式，这里将基本块的划分修改为(以中间代码为例)

label或init\_fun开始

call函数调用结束，函数返回开始

条件跳转beq与无条件跳转j

程序结束exit

经过考虑，同一个基本块会有很多个入口，但对于出口只有顺序执行和跳转两个，否则这个基本块的划分就不完全。

编译执行时，每个函数的第一个和最后一个基本块分别设置为函数入口与出口，入口没有前驱基本块而出口没有后继基本块，这样便于基本块结构的划分以及数据流划分。

算法：

建立基本块的算法主要就是根据不同的操作符进行基本块的划分，首先读取存在内存中的序列，然后根据mid->op进行基本块的划分，遇到条件跳转指令，j指令，call就进行基本块的分割，当前基本块可以结束，同时开辟一个新的基本块，同时条件跳转与call的follow\_block设置为对应新block值，遇到label，fun\_init也是如此开启新的基本块，ret也是如此。此时所有新块就被划分出来了，然后进行第二次扫描进行连接操作，如果当前基本块的j\_name与另一基本块的name相同则进行连接，最终就可以分配好流图。

#### 公共子表达式导出

由于书上的公共子表达式删除支持的指令种类太少，所以我们将它扩充至包含scanf，printf，push上面，同时书上也没有写明白dag图上的变量被kill掉的情况。

1. scanf printf push

scanf相当于对变量的重新定义，与赋值语句别无二致，所以这里就把scanf看作没有src的一个赋值语句，node的操作符设为SCANF，这样相当于会建立一个由空到目标变量的一段dag图，之后再将dag图的名称以及dag\_list上的名称设为对应值即可。

printf与push是对变量的使用，但是这里printf与push有顺序问题，所以必须单拿出一个结点进行存储，所以最后实现时在戴用变量的上方建立新的node，同时将node的op标注为PRINT或PUSH，相当于原样输出这个中间代码。

2.中间结点的名称保留

为每一个中间结点建立一个node\_candidate的set集合，每一次一旦有一个新的结点指向了这个node结点，就把新节点的名称传给该node的node\_candidate，然后node的名称如果是临时变量就尽量变为局部或者全局变量。然后在取用一个结点表的一个结点src时，如果命中，那么要检查一下结点表中的idx(对应dag图中的序数)指示的变量a在结点表中是否有比当前更大的序数，即该变量a是否有新定义值，如果有新定义值的话就要在当前的dagnode的node\_candidate中删去该name，将src的name填入原来a的位置中以防恢复中间代码的时候产生冗余的恢复代码。

3.中间代码的导出

书上采用的是启发式算法，书上采用如此的顾虑是因为编译器的寄存器较少，其实在我们这么多寄存器的前提下，不需要采用启发式算法完全得以应付绝大多数情况，同时还可以将printf与push等按顺序输出而不至于产生错误。

活跃变量分析首先就是对各个函数块中各个基本块的in，out，use，def的分析，in和out是通过进出基本块的流图来确定的，在之前建立的基本块的基础上先确定use/def图

1. 对于四则运算与正负赋值，检查src1，src2可否加入use集合，dst可否加入def集合。

2. 数组给变量赋值，被赋值变量检查是否加入def，数组下标操作数及数组变量检查加入use集合。

3. 变量给数组赋值，检查赋值变量加入use，数组与下表操作数加入def。

4. 条件跳转语句判断操作数是否加入use

5. scanf语句输入变量检查是否加入def集合

6. printf，push语句使用的操作数是否加入use集合。

之后先扫描一遍各函数块的DAG块即可给出对应的use与def集合，再由in与out的书上公式：

in[B] = use[B]∪(out[B]-def[B])

out[B] = ∪B的后继基本块in[F]

来计算各个in与out

#### 着色法对全局寄存器的分配

在活跃变量分析的基础上，定义基本块内变量之间冲突图，这个冲突的定义按书上“两个变量中的一个变量在另一个变量定义(赋值)处是活跃的，它们之间有一条边连接”的工程量有一些大，先保守地取in集与def集同时为1的时候判断该变量为活跃，在一个基本块同时出现活跃的变量为冲突变量，之后执行书上算法15.1的图着色算法即可。在着色时，首先进行结点删除，这时对不可分配的结点的remove置为-1，否则分配的话按照先后顺序从零开始增加，相当于进入队列的顺序，恢复的时候逆序恢复分配即可。

#### 窥孔优化

1.连续的label，若是连续的label中有函数开始名则将这些保存为函数开始名，若是没有函数开始名则将这个保存为最后一个的label。

2.跳转的目标语句还是跳转语句，可以将第后面的跳转语句删除。

3.跳转与对应label直接相连，这是可以直接将j删去

**课设建议：**

编译课设作为北航计算机与软件工程系的一门老牌名课，其地位性自不言而喻。这门课的总体体验还是不错的，和老师和助教一个学期的课程学习体验还是很棒的，尤其是我们班的两个助教总是能够在线上线下都及时回复同学们的问题，对尾门很有帮助，但是在一个学期的学习之后有一些小小的建议想表达一下。

·教师的上课方向侧重点不是特别一样，像隔壁班史老师作为专门测试我们优化的老师，对这部分十分熟稔，在课堂上或多或少会渗透许多编译优化的小技巧，这样可能其他班级的同学就经常费破脑袋想一个问题但是可能就是一句话就解决的事情，所以建议史老师能就编译优化这一块单开出几节课在大教室讲一下，让更多的学生能够在一定的高度上开始思考问题而不是完全地平地起高楼，这样效率就会高很多。

·论坛的提醒机制感觉并不完善，而且各种约定也不是一个在一个清晰可见的明面上，直到最后还经常有冒出来的约定。所以希望老师能够在以往的教学经验上总结出一个确定无疑的约定1.0在开始进行编译的时候就放出来，这样的话学生在实现的时候就会安心地按照准则去做而畏手畏脚，有的时候还和课程组闹出不愉快，只要做到提前沟通，这些小摩擦还是会很早就解决的。

最后还是对帮助过我的所有老师、助教和同学表示由衷的谢意，希望北航的编译课设越做越好，让这门课为学弟学妹带来更多的乐趣！