《编译技术》

程设优化文档

学号：\_\_\_\_\_16061040\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_王科翔\_\_\_\_\_\_\_

2019年1月4日

## 一．引言

编译原理毫无疑问是计算机学院大三学期压力最大的课程。在这门课的投入也是我本学期的投入最多的。在前期的设计阶段，我参考了大二下学期的面向对象的课程，将面向对象课程的基本原理融入这门课，所写出来的程序有较高的可可读性和较低的代码耦合性。凭借较好的面向对象方式，我比较顺利的完成了前两次的测试，即使遇到问题，也能凭借良好的代码风格及时予以修改。在优化阶段，我也尽我所能的进行了优化。后面我将分章节一一讲解我的优化内容。

## 二．常数优化

常数优化是我在完成生成代码任务的时候，直接嵌入生成代码中的，具体方法为：

对于计算式 a = b + c。

如果b和c都是常数，假如a的寄存器是$t3，b和c的和是d，直接使用命令：li $t3 d。

如果其中的一个是常数，也会使用addiu等I型命令，从而减少代码行数。

但是，如果从语法分析的过程就进行处理，理论上是可以在语法分析阶段就将所有常数运算计算完毕。设计上不难实现，但是由于时间不够，没有实现。

## Dag图消除

Dag图的设计比想象得要复杂很多。书上的算法显然是不能应用于所有的场景的，对于函数调用，数组，其中的运算均不是简单的单独的栈可以解决的。

我的设计方案是，利用map容器设计两个追踪列表，一个用于变量追踪，一个用于中间变量，也就是计算的临时结果的追踪。并设置一个追踪函数，可以对一个quantity量，追踪之前计算的结果，返回追踪的结果quantity。

对于每个变量型四元式：追踪变量所赋予的值。将标量压入变量追踪的map

对于每个计算节点四元式：追踪进行运算的两个值。检查追踪列表之中有无可以替换的节点，如果有就替换。

对于每个函数型四元式：追踪所有参数。

对于每个branch型四元式：追踪进行比较的一个或者两个表达式。

对于每个return型四元式：追踪返回的表达式。

对于每个printf型四元式：追踪打印的表达式。

对于每个数组型四元式：追踪位置的表达式和里面的值得表达式，之所以不像变量一样压入map，是因为仅凭靠数组名不能唯一确定一个值。同一个数组的不同位置得值共享同一个名字。

最初的设计是一个map就储存所有的值，包括变量和计算节点。之所以使用两个列表，是因为，不希望进行变量级的追踪以及函数前后的全局变量的追踪无法解绑。比如：

g = a;

a = a +1;

b = g;

进行变量级的追踪会，优化结果为 b = a;但是里面的a 却是新值。

此外由于计算顺序和传参顺序的问题，给每个全局变量都包了一个临时变量，这个变量会被优化掉，产生错误。

而对于全局变量解绑的问题例如：

x = 1 + 0; // x 为全局变量

add(); // add中修改了x的值，例如x = x + 1

printf(x);

输出结果会是2，原因在于运行add结束后，对x追踪的结果仍是1 + 0的结果。对于这种的情况，我专门设计了一个去除全局变量进行追踪的函数。但是下面的情况依旧存在。

x = 1;

d = x + 1;

add();

d = x + 1;

printf(d);

对于这种情况，仍然会输出2，因为，虽然x 的值被去追踪，但是x +1 仍然被追踪。d 会追踪到旧的x + 1。考虑到多层嵌套，一个一个检查下面是否还有全局变量实现难度太高。

基于以上遇到的问题，选择将变量和计算节点分离。所有的变量放一个，计算节点放一个。由于变量本身不具有深度，也就是说变量的名字就可以唯一确定一个quantity量。所以，对于变量的索引直接使用了变量名。

## 四．寄存器分配

对于寄存器的分配我的直接将优化代码写入了代码生成的程序中，根据情况进行动态的分配。具体策略使用的是引用计数。

在对于一个基本块开始生成之前，获得所有变量（包括数组）的使用次数。对于次数进行排序，对于使用次数的最多的变量优先分配给store寄存器。

对于其他的成分，比如计算节点，使用temp寄存器。具体的分配策略就是先进先出。当八个寄存器没满的时候，直接取后面的寄存器。如果寄存器已满，就调用overflow，处理寄存器栈溢出的情况。

改进思路现在的所有寄存器都是使用寄存器名来调用的，例如$t2，这样的对于我的结构想要改成数字的有一定难度，主要在于需要很细致地一个一个改，但是对于我的数据结构只要能把代码中所有的生成相关的代码修改，可以直接适用。由于时间不足，没有进行此项尝试。如果修改，可以更加灵活，减少回写的次数。减少momery操作。

对于store寄存器和temp寄存器均实现了需要多少存多少，减少了momery操作。尤其是temp寄存器由于每次使用之前我都会默认里面是0，而导入参数之后我也会把内容会写。所以当从一个进入一个的函数的时候我不再专门保存一边t寄存器，这极大的减少了代码量。

## 五．感受

由于考试安排的原因，1.4日下午我曾先后参加了数据库和机器学习两个考试，时间非常紧张，很多优化的想法没有来得及实现，优化的文档也不是很完善。我最早写的是寄存器分配，在生成代码是就考虑了寄存器分配，优化的时候反而没有怎么进行修改，一些细节已经遗忘。虽然感觉自己所做的仍有进步空间，但是我也已经尽力了，已经在考期无数次熬夜。